UNIVERSIDAD AUTONOMA DE NUEVO LEON
FACULTAD DE INGENIERIA MECANICA Y ELECTRICA
DIVISION ESTUDIOS DE POST-GRADO

DESARROLLO DE SESIONES DE TRABAJO PARA LA PROGRAMACION DE MICROPROCESADORES

POR
ING. JOSE ANGEL CASTILLO CASTRO

TESIS

En opción al Grado de Maestro en Ciencias de la Ingeniería Eléctrica con especialidad en Electrónica

CD. UNIVERSITARIA DICIEMBRE 1997

CASTILLO CASTRO

DESARROLLO DE SESIONES DE TRABAJO PARA LA PROGRAMACION DE MICROPROCESADORES

1997

UNIVERSIDAD AUTONOMA DE NUEVO LEON

FACULTAD DE INGENIERIA MECANICA Y ELECTRICA

DIVISION DE ESTUDIOS DE POSTGRADO

DESARROLLO DE SESIONES DE TRABAJO PARA LA PROGRAMACION DE MICROPROCESADORES

POR

ING. JOSE ANGEL CASTILLO CASTRO

TESIS

EN OPCION AL GRADO DE MAESTRO EN CIENCIAS DE LA INGENIERIA ELECTRICA CON ESPECIALIDAD EN ELECTRONICA

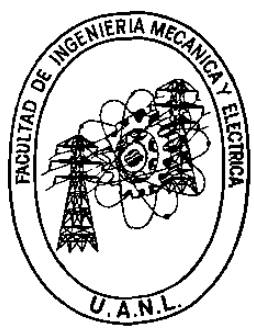
CD. UNIVERSITARIA DICIEMBRE DE 1997

25- Aug. 08
Oklahoma City.

UNIVERSIDAD AUTONOMA DE NUEVO LEON

FACULTAD DE INGENIERIA MECANICA Y ELECTRICA

DIVISION DE ESTUDIOS DE POSTGRADO



DESARROLLO DE SESIONES DE TRABAJO PARA LA PROGRAMACION DE MICROPROCESADORES

POR

ING. JOSE ANGEL CASTILLO CASTRO

TESIS

EN OPCION AL GRADO DE MAESTRO EN CIENCIAS DE LA INGENIERIA ELECTRICA CON ESPECIALIDAD EN ELECTRONICA

CD. UNIVERSITARIA DICIEMBRE DE 1997

UNIVERSIDAD AUTONOMA DE NUEVO LEON
FACULTAD DE INGENIERIA MECANICA Y ELECTRICA
DIVISION DE ESTUDIOS DE POST-GRADO

Los miembros del comité de tesis recomendamos que la tesis **Desarrollo de Sesiones de Trabajo para la Programación de Microprocesadores** realizada por el **Ing. José Angel Castillo Castro** sea aceptada para su defensa como opción al grado de Maestro en Ciencias **de la Ingeniería Eléctrica** con especialidad en **Electrónica**.

El Comité de Tesis

Asesor
M.C. Luis Manuel Camacho Velázquez

Coasesor
M.C. Sergio Martínez Luna

Coasesor
M.C. Roberto Villarreal Garza

Vo.Bo.
M. C. Roberto Villarreal Garza
División de Estudios de Postgrado

San Nicolás de los Garza, N.L. a 5 de Diciembre de 1997.

PROLOGO

Esta tesis fue elaborada en un trabajo conjunto entre el Ing. José Angel Castillo Castro y el Ing. César A. Leal Chapa, ambos profesores de la materia de Electrónica Lógica III (Diseño con Microprocesadores) que se imparte en las carreras de Ingeniero en Control y Computación e Ingeniero en Electrónica y Comunicaciones de la Facultad de Ingeniería Mecánica Eléctrica de la UANL.

El propósito de desarrollar esta Tesis es la de aportar un documento que contenga los temas mas importantes para la enseñanza y aprendizaje de microprocesadores y los programas de cada tema discutido, que sirva como guía y apoyo para que el alumno implemente en la práctica los conceptos de programación y diseño adquiridos en el salón de clase.

Los temas fueron desarrollados alrededor del Microcontrolador MC68HC811E2, dispositivo que se trata en el programa de clase, aunque en algunos capítulos los temas son desarrollados en una forma general sin especificar algun microprocesador en especial, esto con la intención de que sirvan además como una base teórica y una consulta bibliográfica para el aprendizaje de técnicas de programación con cualquier microprocesador.

TABLA DE CONTENIDO

PROLOGO i

INDICE ii

LISTA DE ABREVIATURAS v

LISTA DE FIGURAS viii

LISTA DE TABLAS xi

RESUMEN xii

1. Introducción 1

1.1 Antecedentes 1

1.2 Fundamentos a la metodología propuesta 1

2. Descripción del PCBUG (Práctica No. 1) 3

2.1 Instalación del PCBUG11 3

2.2 Iniciando el PCBUG11 4

2.3 Descripción de la Pantalla 5

2.4 Comandos mas utilados 6

2.5 Procedimiento para la prueba de un Programa 7

2.5.1 Ensamblador en línea, Comando ASM 8
2.5.2 Lenguaje Ensamblador CROSS (C16) 10

3. Modos de Direccionamiento (Práctica No. 2) 13

3.1 Modo Inherente 14
3.2 Modo Inmediato 15
3.3 Modo Directo 16
3.4 Modo Extendido 18
3.5 Modo Indexado 19
3.6 Modo Relativo 21

4. Estrategias para el Diseño de Programas (Práctica No. 3) 24

4.1 Lazos Iterativos 24
4.2 Serie de Números 28
4.3 Lazos de Retardo de Tiempo 31
4.4 Tabla de Datos 35

4.4.1 Solución de Algoritmos Númericos mediante una Tabla de Datos . . 35
4.4.2 Manipulación de datos mediante una Tabla 42

5. Puerto Paralelo (Práctica No. 4) 47

6. Puerto Serie (Práctica No. 5) 51

7. Interrupciones (Práctica No. 6) 55

7.1 Reset 56
7.2 Interrupciones por Hardware y Software 57

8. Otros bloques del 68HC11 (Práctica No. 7) 59

8.1 EEPROM 60
8.2 Convertidor Análogo-Digital 63

9. Aplicaciones (Práctica No. 8) 66

9.1 Sesion 1 Controlador Teclado - Display 67

9.1.1 Descripción 67

9.1.2 Metodología 67

9.1.2.1 Display Multidígito 68
9.12.2 Programa Encodificador de Teclado 70
9.1.2.3 Controlador Display-Teclado 72

9.2 Sesion 2 Motor de Pasos (Posicionador) 73

9.2.1 Descripción 73
9.2.2 Metodología 73

CONCLUSIONES Y RECOMENDACIONES 77

REFERENCIAS 78

GLOSARIO 80

RESUMEN AUTOBIOGRAFICO 82

LISTA DE ABREVIATURAS

ASM	Ensamblador en linea (Comando del PCBU11)
ABA	Suma el Acumulador A con el Acumulador B (Add Acunulator A to B)
ADDA	Suma sin accrreo el Acumulador A (Add without carry)
BCD	Binario Codificado en Decimal (Binary Coded Decimal)
BCS	Brinca si hubo acarreo (Branch if carry set)
BEQ	Brinca si es igual (Branch if equal)
BNE	Brinca si no es igual (Branch if not equal to zero)
BPROT	Protección de Bloque (Block Protection)
BRA	Brinco incondicionado (Branch always)
CD	Cambia directorio (Comando del DOS)
CLR	Borra contenido de memoria (Clear)
C16	Cross Asssembler 16 (Programa Ensamblador C16)
DEC	Decrementa memoria (Decrement)
DECA	Decrementa acumulador A (Decrement to acumalator A)
DECB	Decrementa acumulador B (Decrement to acumalator B)
DOS	Sistema Operativo de Disco (Disk Operative System)
EEPROM	Memoria de Sólo Lectura Borrable Eléctricamente Programable (Erasable Electrically Programmable Read Only Memory)

LISTA DE ABREVIATURAS (CONTINUACION)

HOF	Archivo de Hexadecimal de Salida (Hexadecimal Output File)
LED	Diodo Emisor de Luz (Ligth Emitter Diode)
INC	Incrementa memoria (Increment)
INCA	Incrementa Acumulador A (Increment Acumulator A)
INCB	Incrementa acumulador B (Increment Acumulator B)
LDAA	Carga Acumulador A (Load Acumulator A)
LDAB	Carga Acumulador B (Load Acumulator B)
LDX	Carga Registro Indice X (Load Index Register X)
LDY	Carga Registro Indice Y (Load Index Register Y)
LSI	Gran Escala de Integración (Large Scale Integration)
MCU	Unidad de Microcontrolador (Microcontroller Unit)
MD	Despliega Memoria (Comando del PCBUG11)
MM	Modifica Memoria (Comando del PCBUG11)
MPU	Unidad de Microprocesamiento (Microprocessor Unit)
MSI	Mediana Escala de Integración (Medium Scale Integration)
NRZ	No Retorno a Cero, Estándar de señal eléctrica en comunicación serie (Non Retur Zero)
ORG	Origen (Origin)

LISTA DE ABREVIATURAS (CONTINUACION)

PA	Puerto A
PB	Puerto B
PC	Puerto C
PD	Puerto D
PE	Puerto E
RAM	Memoria Vólatil de Acceso Aleatorio (Random Access Memory)
RD	Despliega Registro, Comando del PCBUG11 (Register Display)
RM	Modifica Registro, Comando del PCBUG11 (Register modify)
ROM	Memoria de Sólo Lectura (Read Only Memory)
RTS	Retorno de Subrutina (Return to Subroutine)
SCI	Interfase de Comunicación Serial
SPI	Interfase de Perifericos Síncronos
STAA	Almacena Acumulador A (Store Acumulator A)
SWI	Interrupción por Software (Software Interruption)
TERM	Terminal (Comando del PCBUG11)
UART	Receptor/Tansmisor Universal Asíncrono (Universal Asynchronus Receiver/Transmitter)

LISTA DE FIGURAS

FIGURA	DESCRIPCION	PAGINA
2.1	Registro BPROT para protección del EEPROM	8
3.1	Programa ejemplo del modo de direccionamiento inherente	14
3.2	Programa que incrementa el Acumulador A	14
3.3	Programa que incrementa el Registro Ingice X	15
3.4	Estructura de una instrucción del modo inmediato	15
3.5	Programa que carga el Acumulador A en modo inmediato	16
3.6	Programa que carga el Registro X en modo inmediato	16
3.7	Estructura de una instrucción del modo directo	17
3.8	Programa que carga el Acumulador A en modo directo	17
3.9	Programa que almacena el Acumulador A en modo directo	18
3.10	Estructura de una instrucción del modo extendido	18
3.11	Programa que carga el Acumulador A en modo extendido	19
3.12	Programa que almacena el Acumulador A en modo extendido	19
3.13	Estructura de una instrucción del modo indexado	20
3.14	Programa que carga el Acumulador A en modo indexado X	20
3.15	Programa que almacena el Acumulador A en modo indexado X	21
3.16	Estructura de una instrucción del modo relativo	21

LISTA DE FIGURAS (CONTINUACION)

FIGURA	DESCRIPCION	PAGINA
3.17	Programa ejemplo para verificar el cálculo adecuado del OFFSET	22
3.18	Programa para prueba del modo relativo	23
4.1	Diagrama de flujo de un programa de lazo	25
4.2	Diagrama de ideas de un programa que cuenta los bits con valor 1 de la dirección 0050	26
4.3	Programa que cuenta los bits con valor 1 de la dirección 0050	27
5.1	Conexión de un LED para controlarse por el puerto B de salida	47
5.2	Conexión de un interruptor para sensarse por el puerto C de entrada	49
5.3	Operadores LógicosAND, OR, OR	50
6.1	Programa de transmision de datos	53
7.1	Conexiones de interrupciones externas al MPU	56
7.2	Secuencia de inicio de un Microprocesador	57
7.3	Diagrama de flujo de secuencia de interrupción	58
8.1	Registro de Control de Programación del EEPROM	60
8.2	Diagrama de Bloques del Convetidor Análogo/Digital	57
8.3	Registro ADTL para opración del Convertidor A/D	57
8.4	Registro OPTION, bit ADPU	58

LISTA DE FIGURAS (CONTINUACION)

FIGURA	DESCRIPCION	PAGINA
9.1	Diagrama a bloques del Módulo Display-Teclado	67
9.2	Diagrama a bloques de un Controlador de un Display Multidígito	68
9.3	Diagrama de ideas de un programa controlador de display multidígito	63
9.4	Diagrama de ideas de un programa encodificador de teclado matricial	64
9.5	Diagrama a bloques de un Módulo de Motor de Pasos	67
9.6	Diagrama de ideas de un programa para el avance de un paso	68
9.7	Diagrama de ideas de una subrutina de avance de n pasos	69

LISTA DE TABLAS

TABLA	DESCRIPCION	PAGINA
2.1	Asignación de bits del Registro BPROT para protección del EEPROM	8
4.1	Ejemplo de tabla de datos	35
5.1	Puertos de Entrada/Salida del Microcontrolador MC68HC811E2	42
5.2	Lógica de operación de un LED controlado por el Puerto B de salida	43
5.3	Logica de operación de un interruptor sensado por el Puerto C de entrada	44
6.1	Registros del SCI	47
8.1	Opciones de borrado del EEPROM	55
9.1	Asignación de lineas del Puerto A para el control del Motor de Pasos	68
9.2	Convención de sentido de giro para el motor de pasos (Programa 1)	70

RESUMEN

La enseñanza y el aprendizaje del diseño de sistemas electrónicos de tipo digital basado en microprocesadores, requiere de una intensiva aplicación del conocimiento teórico.

La evaluación del conocimiento de un alumno, es contundente cuando éste es capaz de comprobar en la práctica que los programas que desarrolla operan adecuadamente.

Además, la forma más adecuada en que un alumno puede verificar si el programa que ha desarrollado es correcto, es comprobar su operación en forma práctica.

En esta tesis se abordan los temas mas importantes en la enseñaza y aprendizaje de microprocesadores. En el Capítulo 2 se describe el Sistema de Evaluación PCBU11, así como los comandos más comunes y se concluye con la descripción paso a paso para la evaluación de un programa. En el captulo 3 se ilustran los modos de direccionamiento asi como ejemplo sencillos para la comprobación de los mismos. Las técnicas de programación y estructuras típicas de los programas son discutidas ampliamente en el Capítulo 4. Los Dispositivos de Entrada/Salida tipo Paralelo y tipo Serie son descritos en los Capítulos 5 y 6 respectivamente. Las condiciones de excepción del Microprocesador como los son el RESET y las Interrupciones se detallan en le Capítulo 7, El estudio de la memoria tipo EEPROM y el convertidor Analógo-Digital se ilustra en el Capítulo 8, el desarrollo de un par de aplicaciones típicas; el Control de un Teclado-Display Multidigito y el Control de un Motor de Pasos se llevan a cabo el Capítulo 9.

CAPITULO 1

Introducción

1.1 ANTECEDENTES

Uno de los problemas básicos en la enseñanza de microprocesadores es que ha sido empleado ampliamente el método tradicional de exposición descriptiva, con resultados poco favorables, ya que se desvincula la teoría de la aplicación del conocimiento.

En algunos casos se intenta apoyar a la discusión teórica con modernos sistemas computarizados de simulación absoluta, con lo que se aleja más al alumno de la aplicación real que es el objetivo final del entrenamiento en esta área.

El proceso de enseñanza/aprendizaje de microprocesadores debe cubrir cuando menos las siguientes 3 aspectos fundamentales.

a) Favorecer la comprensión y asimilación del conocimiento teórico.
b) Facilitar el desarrollo de la habilidad abstracta de crear programas.
c) Desarrollar la habilidad psico-motora o manual para probar en la práctica el programa y un prototipo.

1.2 FUNDAMENTO DE LA METODOLOGIA PROPUESTA

El fundamento de esta tesis radica básicamente en la aplicación del conocimiento en forma inmediata, a través del diseño adecuado de experimentos y el apoyo de un sistema

de desarrollo que cuente con los modelos de hardware para efectuar situaciones reales en casos de aplicación típica.

Para lo cual se han desarrollado las discusiones de los temas más importantes, integrando experimentos para su comprobación y dejando propuestas de problemas para el alumno.

El desarrollo de esta tesis fue efectuado en el siguiente orden:

En el Capítulo 2 se hace una descripción del PCBUG11, herramienta para el desarrollo de programas a partir de una computadora personal.

Los modos de direccionamiento se discuten en el Capítulo 3 , forma básica de la clasificación de instrucciones, así como el conjunto de instrucciones de un microprocesador.

Las estrategias para el diseño de programas se tratan en el Capítulo 4, donde además se proponen modelos típicos para la creación de programas.

El puerto de tipo paralelo y el puerto serie son tratados en los Capítuos 5 y 6 respectivamente.

El tema de interrupciones y el procedimiento para convertir al microprocesador en dispositivo autónomo se discute en el Capítulo 7.

En el Capítulo 8 se describen otros bloques contenidos en el paquete 68HC11.

La presentación de aplicaciones típicas y su procedimiento de diseño se llevan a cabo en el Capítulo 9 .

CAPITULO 2

Descripción del PCBUG11
(Práctica No. 1)

OBJETIVO

El objetivo de este tema es el de presentar y describir el Sistema de Evaluación PCBUG11 como herramienta de desarrollo de aplicaciones con Microprocesadores.

INTRODUCCION

El PCBUG11 es un Sistema de Evaluación para los Microcontroladores de la Familia HC11. Este sistema se compone de 2 partes; un programa que es ejecutado en una computadora personal en ambiente DOS, que tiene como acción inicial la de transferir un programa a la memoria del MPU, que es la otra parte del sistema y consiste en un programa llamado Programa Monitor , estos dos programas interactúan a través del puerto serie para conformar un sistema de evaluación que permite al usuario desarrollar sus programas de prueba o como parte de una aplicación mediante comandos que permiten introducir programas y/o datos a los dispositivos de memoria, a los registros del MPU y a los registros de control de dispositivos de entrada/salida, asi como el de ejecutar los programas.

2.1 INSTALACIÓN DEL PCBUG11

Motorola proveé este paquete de programas en un disco flexible, el cual se debe de instalar en el disco duro de la computadora. Opcionalmente se puede ejecutar desde el disco flexible.

Los pasos para instalarse en el disco duro, son los siguientes:

1.- Insertar el disco flexible en el drive A.

2.- Crear un subdirectorio

md PCBUG11

3.- Abrir el nuevo directorio

cd PCBUG11

4.- Copiar todos los archivos del disco flexible al diso duro

copy a:*.* c:

2.2 INICIANDO EL PCBUG11

Para iniciar, una vez conectada la tarjeta del microcontrolador a la computadora a través de los puertos de comunicación (COM1, COM2), es necesario ejecutar el programa PCBUG11, adicional a esto es necesario indicar varios parámetros, tales como el puerto, la frecuencia del cristal, la versión del MCU, etc.

Si se introduce solamente PCBUG11, se ejecutará un programa previo que cuestionará al usuario sobre las condiciones del hardware para así establecer los parámetros adecuados.

Ejemplo.-

Si se emplea un MC68HC811E2, a través del COM2 con un cristal de 8 MHZ.

La opción será: PCBUG11 -A port=2

Una vez que se han determinado los parámetros, se tiene enseguida la opción de ejecutar el programa monitor.

Se recomienda crear un archivo PCBUG.bat que sea ejecutado desde el directorio raiz y que contenga los siguientes comandos:

cd PCBUG11

PCBUGII [Parámetro MCU] [Parámetro puerto] [Parámetro baud]

Para así tener un procedimiento de inicio rápido.

2.3 DESCRIPCIÓN DE LA PANTALLA

El PCBUG11 muestra a través del monitor de la computadora el estado de la electrónica asociada con el MCU (registros, bloques de memoria, vectores, etc), además del resultado de los diferentes comandos, para esto la pantalla del monitor se divide en varias ventanas:

Para la descripción de cada pantalla se indica el color con el que se visualizará cada una de éstas, si acaso se cuenta con un monitor a color.

Ventana principal.- Esta ventana ocupa al mitad más alta de la pantalla, con textos en color blanco con fondo azul, despliega el resultado de los comandos y el contenido de la memoria.

Ventana de registros.- Se encuentra al centro de la pantalla, con los textos en color blanco o amarillo y con el fondo rojo. Muestra el contenido de los registros del MPU.

Ventana de estado.- Se encuentra al centro y a la derecha, con textos en blanco y fondo morado. Muestra el MCU en uso (Al, 811E2, etc.), y el estado del MCU: ejecutando programa (running), punto de quiebre (break), paso por paso (tracing).

Ventana de comando.- Esta situada en la parte baja izquierda de la pantalla, es de color negro con textos en blanco. Se usa para introducir y leer los comandos del PCBUG11. El caracter ">>" aparecerá siempre al inicio de la línea de comandos.

Existen dos ventanas temporales que aparecerán sobrepuestas en la pantalla principal.

Ventana de error.- Está ventana indica cualquier error o una operación incorrecta en las comunicaciones hacia el MCU. Para borrar esta pantalla, se presiona cualquier tecla o se espera 5 segundos y la pantalla desaparecerá por sí misma.

Ventana de ayuda.- Esta ventana muestra ayudas para el uso de los comandos del PCBUG11.

2.4 COMANDOS MAS UTILIZADOS

Para la evaluación y/o el desarrollo con un microcontrolador se requieren de comandos que efectúen básicamente tres acciones:

1.- Introducir en memoria los códigos de instrucción y/o datos previos.

2.- Visualizar los registros del MPU o alguna memoria para verificar el buen funcionamiento del programa.

3.- Ejecutar los programas

En esta sección se describen los comandos que se usan con mas frecuencia, en la descripción de cada comando se indica la sintaxis del mismo así como el ejemplo, además el PCBUG11 cuenta con una ayuda -Comando HELP- en la que se describe en detalle la totalidad de los comandos

MD.- Desplegar memoria (Memory Display).- Este comando despliega el contenido de la memoria, junto con el comando se debe indicar el inicio y el final del rango de memoria, si solamente se indica la dirección de inicio, el PCBUG11 despliega solo 16 direcciones.

MD [Dir. inicio] [Dir. final]

Ejemplo:

MD $F800 $F9FF Despliega el contenido de las direcciones $F800 a $F9FF

MD$1000 Despliega el contenido de la dirección $1000 a la $100F

MM.- Modificar memoria (Memory modify).- Este comando sirve para modificar el contenido de la memoria.

Sintaxis: MM [Dirección]

Después de introducir un valor, presione la tecla enter y avanzará a la siguiente dirección desplegando el contenido de ésta.Se puede avanzar o retroceder de dirección con las teclas de flechas, mostrándose siempre el valor actual de la dirección seleccionada.

ASM.- Ensamblador (Assembler).- El PCBUG11 proporciona un ensamblador en línea que permite introducir los programas usando los mnemónicos. Para activar el ensamblador, en la línea de comandos se introduce el programa, para abandonar la función de ensamblador se presiona la tecla Esc.

ASM [Dirección] .

RD.- Despliega Registros (Register Display).- Este comando despliega los valores actuales de los registros del MPU del usuario.

Sintaxis: RD

RM.- Modifica registro (Register Modify).- Este comando permite modificar cualquiera de los registros del MPU de usuario.

Sintaxis: RM

2.5 PROCEDIMIENTO PARA LA PRUEBA DE UN PROGRAMA

En esta sección se describirá el procedimiento para evaluar un programa desde el momento de introducir los códigos a la memoria hasta la verificación de los resultados. Para no aumentar el grado de dificultad que implicaria la implementación de este procedimiento con un programa complicado, se tomará como ejemplo un algoritmo sencillo

que consiste en cargar valores conocidos al acumulador A y B y despues sumarlos.

Los códigos de instrucción se pueden introducir ya sea usando el ensamblador de línea que provee el PCBUG11 mediante el comando ASM, o bien generando los códigos previamente utilizando un Programa Ensamblador, primeramente se describira el procedimiento si se utiliza el Comando ASM.

2.5.1 ENSAMBLADOR EN LÍNEA, COMANDO ASM

1.- Debido a que el PCBUG11 no deja suficientes localidades de Memoria tipo RAM para el usuario, el programa se introducirá en el EEPROM que se encuentra en en la dirección $F800 a la $FFFF. Una vez dentro del PCBUG11 se debe eliminar la protección contra escritura del EEPROM, esta protección se efectúa mediante el registro BPROT ($1035) cuya asignación de bits es la siguiente:

7	6	5	4	3	2	1	0
0	0	0	PTCON	BPRT3	BPRT2	BPRT1	BPRT0
RESET							
0	0	0	1	1	1	1	1

Fig. 2.1 Registro BPROT para protección del EEPROM.

Cada bit BPTRx protege a cada uno de los cuatro grupos de 512 byes del EEPROM de acuerdo con la siguiente tabla.

Bit	Bloque protegido	Tamaño del bloque
BPTR0	$F800-$F9FF	512 Bytes
BPTR1	$FA00-$FBFF	512 Bytes
BPTR2	$FC00-$FDFF	512 Bytes
BPTR3	$FE00-$FFFF	512 Bytes

Tabla 2.1 Asignación de bits del registro BPROT para protección del EEPROM.

De acuerdo con esta tabla, se escribirá un $10 en la $1035 utilizando el comando MM (modifica memoria), de esta manera se habilatará los 2K Bytes del EEPROM.

2.-Como siguiente paso se ejecutará el comando EEPROM, cuya función es la de activar un algoritmo de escritura que provee el PCBUG11 que declara a la memoria tipo EEPROM como memoria de escritura y permite al usuario utilizar a este dispositivo como si fuera del tipo RAM.

El comando EEPROM se utiliza de la siguiente manera:

>EEPROM (direccion inicial) (dirección final)

y para nuestro ejemplo:

>EEPROM $F800 $FFFF

3.- El siguiente paso es el de introducir los códigos que conforman el programa ejemplo. Utilizando el comando ASM:

>ASM $F800(Enter)

Ya dentro de la pantalla principal, se debe de capturar el siguiente listado en mnemónicos, cuidando siempre que después del caracter ">" se debe dejar cuando menos un espacio.

```
$F800 > LDAA   #$5F   ; carga el acumulador A con $5F en modo
                      inmediato
$F802 > LDAB   #$32   ; carga el acumulador B con $32 en modo
                      inmediato
$F804 > ABA           ; Suma el acumulador A con el acumulador B
```

Al terminar de capturar oprimir ESC para regresar a la pantalla de comandos.

4.- Una vez capturados los códigos debe de programarse un punto de quiebre con el comando BREAK, en este caso en la $F805 que es la siguiente localidad de memoria a la última instrucción del programa.

>BREAK $F805(Enter)

5.- Ejecutar el programa con el comando G

>G $F800

6.- Una vez que se ejecute el programa, se observará en la pantalla de estatus la indicación de que el MCU se encuentra en estado de BREAK o punto de quiebre.

para verificar el resultado se ejecutará el comando Desplegar Registros (RD) con esto se actualizará la pantalla de registros con los siguientes valores:

				SXHINZVC
PC $F805	ACCA $91	ACCB $32	CCR $68	%0 1 1 0 1 0 0 0

PC= $F805 Dirección donde se programó el punto de quiebre

ACCA= $91 Resultado de la suma de $5F + $32

ACCB= $32 Valor que se le asignó al acumulador B con la instrucción LDAB

CCR= $68 La suma resultó un valor negativo N=1, ocurrió un medio acarreo H=1 y la máscara de interrupción esta puesto por reset X=1

2.5.2 LENGUAJE ENSAMBLADOR CROSS16 (C16)

1.- Cuando los códigos son generados con un programa ensamblador es necesario elaborar un archivo fuente que contenga el listado del programa en mnemónicos es decir en lenguaje ensamblador, para esto se puede utilizar cualquir procesador de textos, se recomienda usar la utileria de Notes del programa Sidekick ya que este reside en la memoria de la computadora, hecho que facilita las posibles correcciones del programa sin necesidad de abandonar el PCBUG11, a este archivo se le asignará el nombre de SUMA.ASM cuyo contenido será el siguiente:

```
CPU          "6811.TBL"  ;Indicación del archivo que contiene la tabla de
                         mnemónicos y su correspondiente código en
                         lenguaje máquina.
```

```
HOF     "MOT8 "   ;Directivo para indicar el formato del archivo de
                  salida, en este caso formato S19 para
                  Microprocesadores de Motorla de 8 Bits
ORG     0F800H    ;Dirección de inicio del programa
LDAA    #5FH      ; carga el acumulador A con $5F en modo
                  inmediato
LDAB    #32H      ; carga el acumulador B con $32 en modo
                  inmediato
ABA               ; Suma el acumulador A con el acumulador B
```

Cabe mencionar que en este ensamblador los números en base hexadecimal se indican con una letra H al final del número, a diferencia del ensamblador del PCBUG11 en el que se indican con el caracter $ al inicio del valor hexadecimal, además es importante aclarar que en el C16 los valores númericos deben iniciar con un número que podria no ser el caso de un valor base 16, es por eso que en el directivo ORG la dirección se indica 0F800.

2.- El siguiente paso consiste en ensamblar el programa fuente. La función de un programa ensamblador es la de generar a partir del programa fuente un archivo ejecutable que contiene los códigos máquina que corresponder al programa a evaluar.

Dentro del DOS introducir la siguiente línea de comando:

C:\C16 SUMA.ASM [-L SUMA.LIS] [-H SUMA.S19]

Donde:

C16:	Comando para ejecutar el programa ensamblador CROSS16
SUMA.ASM:	Programa fuente
[-L SUMA.LIS]:	Indicación opcional para generar un archivo tipo texto que contiene el listado del programa tanto en mnemónicos como en lenguaje máquina, este archivo es útil para efectos de documentación.

[-H SUMA.S19]: Indicacación para generar el archivo ejecutable, si se omite este parámerto el ensamblador sólo revisará la sintáxis del programa fuente e indicará los errores si éstos existen. Las extensión S19 es opcional, aunque se recomienda usarla para recordar el formato en el que fue ensamblado.

Al terminar el proceso de ensamble se indicará si se detectaron errores en la sintaxis del programa fuente, si no fue asi, el programa esta listo para cargarse a la memoria del MCU.

3.- Antes de cargar el programa es necesario declarar el EEPROM como memoria de escritura, para esto se debe de seguir el procedimiento descrito en el paso 1 y 2 de la sección 2.5.1.

4.- Para cargar el programa se utiliza el comando LOADS, cuya función es la de transferir archivos en formato S19 contenidos en disco hacia la memoria del MCU, ya sea ésta del tipo RAM, EPROM o EEPROM, en estas dos últimas es necesario ejecutar el comando que activa el algoritmo de escritura correspondiente.

En la línea de comandos del PCBUG11 introducir lo siguiente:
>LOADS SUMA.S19 [Enter]
en la pantalla principal se indicará los bytes transferidos.

5.- Lo que resta del procedimiento de prueba se describe en los pasos 4, 5 y 6 de la sección 2.5.1, es decir lo que corresponde a establecer el punto de quiebre o BREAK, ejecutar el programa y verificar el resultado.

CAPITULO 3

Modos de Direccionamiento
(Práctica No. 2)

OBJETIVO

Apoyar el estudio de los modos de direccionamiento, mediante la elaboración de pequeños programas que ejecuten instrucciones en cada modo.

INTRODUCCION

Se le llama modo de direccionamiento a una clasificación de las instrucciones de un microprocesador, con respecto al modo en que se relaciona la instrucción con los registros internos del MPU con los bloques de memoria o con los dispositivos de entrada-salida.

Para los microprocesadores de la familia HC11 los modos de direccionamiento son:

- inherente
- inmediato
- directo
- extendido
- indexado
- relativo

3.1 MODO INHERENTE

En este modo la instrucción está formada por un solo byte, solo el código de instrucción y 2 bytes para instrucciones relacionadas con el registro indice "Y". En este modo las instrucciones actúan solo con los registros internos del MPU.

EJEMPLO.- 1

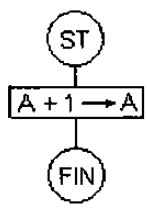
Instrucción - incrementa el acumulador A

Código de instrucción - 4C

Mnemónico - INCA

PROGRAMA:

a) Diagrama de flujo



b) Listado

```
INCA  ;Incrementa el acumulador A
SWI   ;Interrupción por software
```

c) Estatus

Antes de la ejecución del programa

A | nn

Después

A | nn + 1

Fig. 3.1 Programa ejemplo del modo inherente.

EJEMPLO.- 2

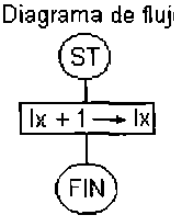
Instrucción - incrementa el registro indice X

Código de instrucción - 08

Mnemónico - INX

PROGRAMA:

a) Diagrama de flujo



b) Listado

```
INX   ;Incrementa el registro índice X
SWI   ;Interrupción por software
```

c) Estatus

Antes de la ejecución del programa

Ix | nn

Después

Ix | nn + 1

Fig. 3.2 Programa que incrementa el registro indice X

EJEMPLO.- 3

Instrucción - incrementa el registro indice Y

Código de instrucción -18 08

Mnemónico - INY

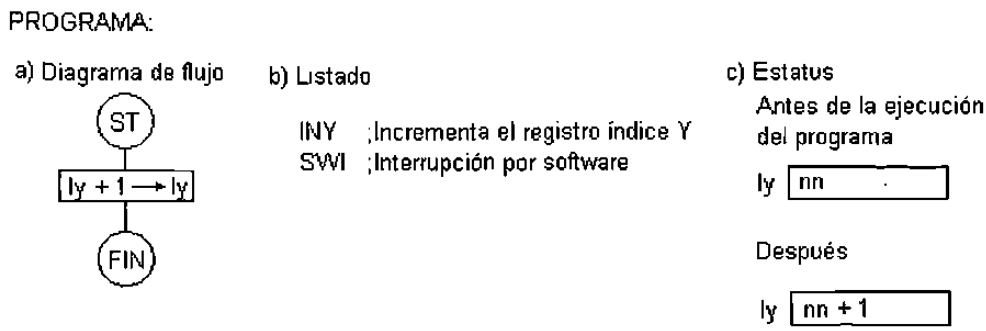


Fig. 3.3 Programa que incrementa el registro índice Y

3.2 MODO INMEDIATO

En este modo la instrucción está formada por 2 bytes, uno para el código de instrucción y otro para el operando; para las instrucciones relacionadas con el registro indice "Y" se requieren 3 bytes. El nombre de inmediato significa que el operando está inmediatamente después del código de instrucción.

Fig. 3.4 Estructura de una instrucción del modo inmediato

EJEMPLO.- 4

Cargar el acumulador A con el número hexadecimal 3A

Instrucción - carga el acumulador A en modo inmediato

Código de instrucción - 86

Mnemónico - LDAA

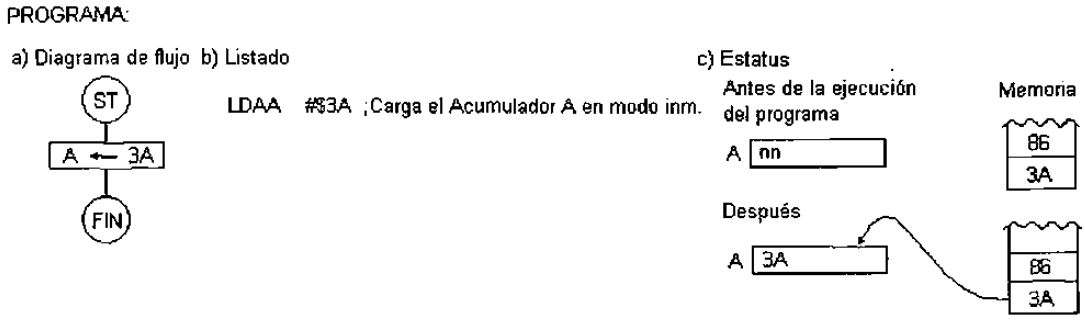


Fig. 3.5 Programa que carga el acumulador A en modo inmediato.

EJEMPLO.- 5

Cargar el registro indice X con FFFF

Instrucción - carga el registro indice en modo inmediato

Código de instrucción - CE

Mnemónico - LDX

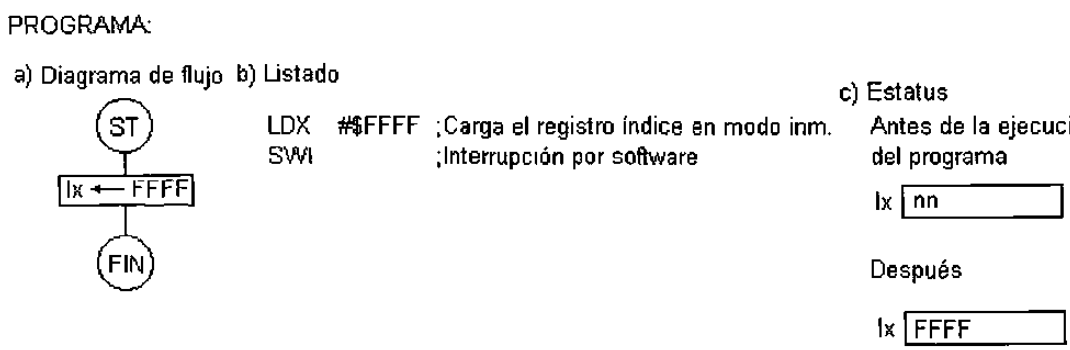


Fig. 3.6 Programa que carga al registro índice X en modo inmediato

3.3 MODO DIRECTO

En este modo la instrucción está formada por 2 bytes, uno para el código de instrucción y otro para indicar la localización del operando; para instrucciones relacionadas con el registro indice "Y" se requieren 3 bytes.

Fig. 3.7 Estructura de una instrucción del modo directo

Al indicar la dirección del operando sólo con un byte, se restringe el uso de este modo a la parte más baja de memoria de 0000 a 00FF, es decir, los primeros 256 bytes.

EJEMPLO.-6

Cargar el acumulador A con el contenido de la localidad 0050

Instrucción - carga el acumulador A en modo directo

Código de instrucción - 96

Mnemónico - LDAA

EJEMPLO.- 7

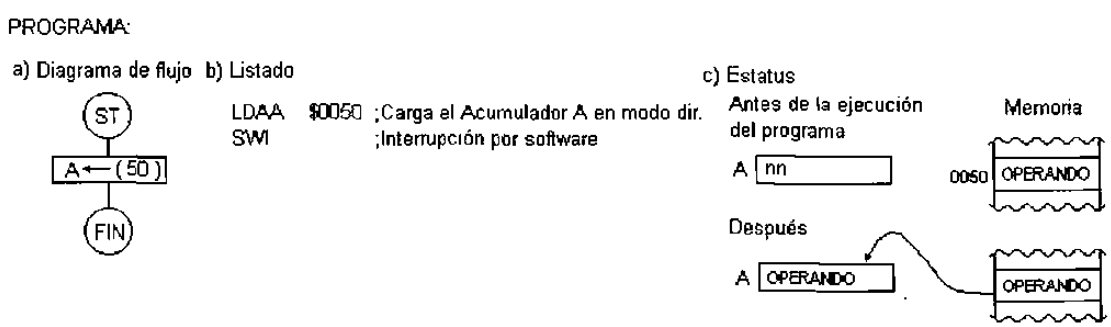


Fig. 3.8 Programa que carga al Acumulador A en modo directo

Depositar el contenido del acumulador A en la localidad 0051

Instrucción - almacena el contenido del acumulador A

Código de instrucción - 97

Mnemónico - STAA DIR

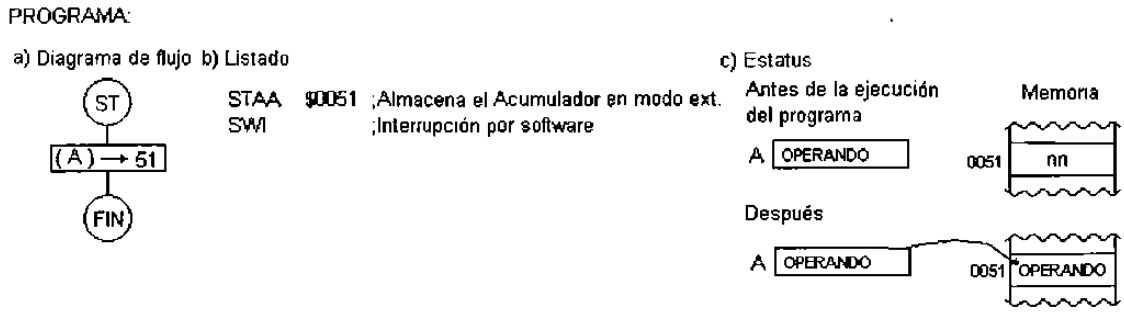


Fig. 3.9 Programa que almacena el Acumulador A en modo directo

3.4 MODO EXTENDIDO

En este modo la instrucción está formada por 3 bytes, uno para el código de instrucción y los 2 restantes para indicar la dirección del operando; para instrucciones relacionadas con el registro indice "Y" se requieren 4 bytes. El modo extendido es una extensión del modo directo ya que al usar 2 bytes para indicar la dirección se puede tener acceso a todo el mapa de memoria de 0000 a FFFF.

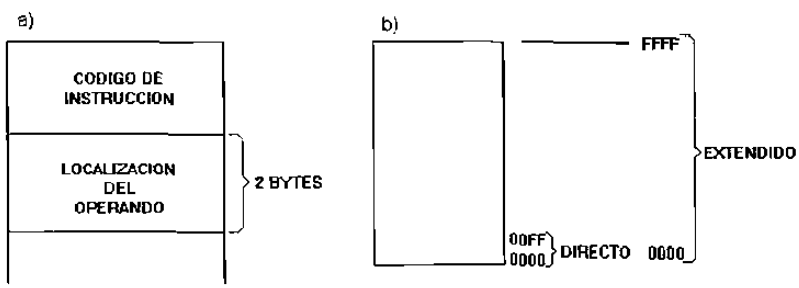


Fig 3.10 a) Estructura de una instrucción del modo extendido
b) Rango de operación

EJEMPLO.- 8

Cargar el acumulador A con el contenido de la localidad F800

Instrucción - carga el acumulador A en modo extendido

Código de instrucción -B6

Mnemónico - LDAA

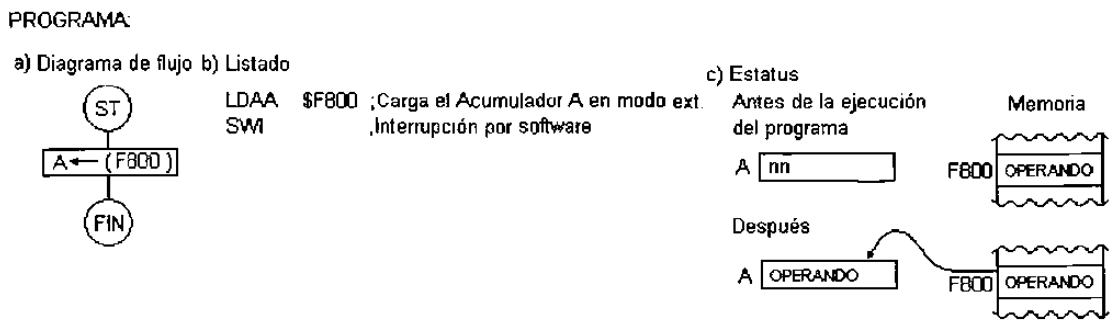


Fig. 3.11 Programa que carga al acumulador A en modo extendido.

EJEMPLO.- 9

Depositar el contenido del acumulador A en la localidad F801

Instrucción - almacena el contenido del acumulador A

Código de instrucción -B7

Mnemónico - STAA

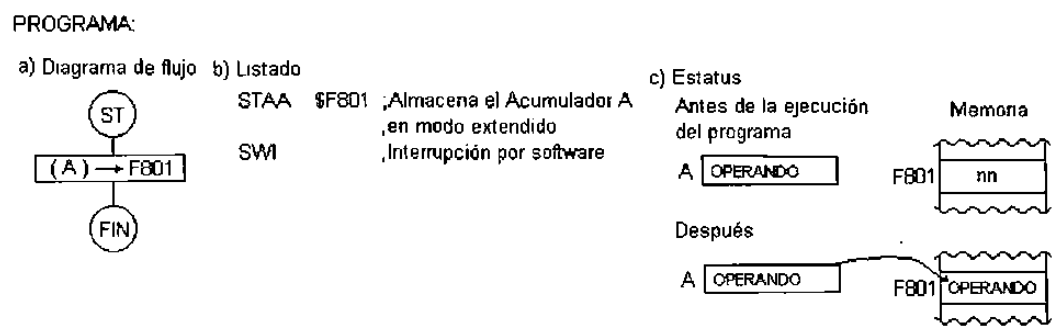


Fig. 3.12 Programa que almacena el acumulador A en modo extendido.

3.5 MODO INDEXADO

En este modo la instrucción está formada por 2 bytes, uno para indicar el código de instrucción y el siguiente para indicar un número de 8 bits que en forma automática se sumará al contenido del registro indice para formar la dirección del operando, a este número se le llama offset y puede tomar un valor de 00 a FF. Para instrucciones relacionadas con el Registro Indice Y se requieren de 2 bytes.

Fig 3.13 Estructura de una instrucción del modo indexado

Dirección del operando = (Ix + OFFSET)

Para usar este modo es necesario haber cargado previamente el registro índice con un valor conocido para formar posteriormente la dirección del operando.

EJEMPLO.- 10

Usando el modo indexado, cargar el acumulador A con el contenido de la localidad 0000

Instrucción - carga el acumulador A en modo indexado

Código de instrucción -A6

Mnemónico - LDAA

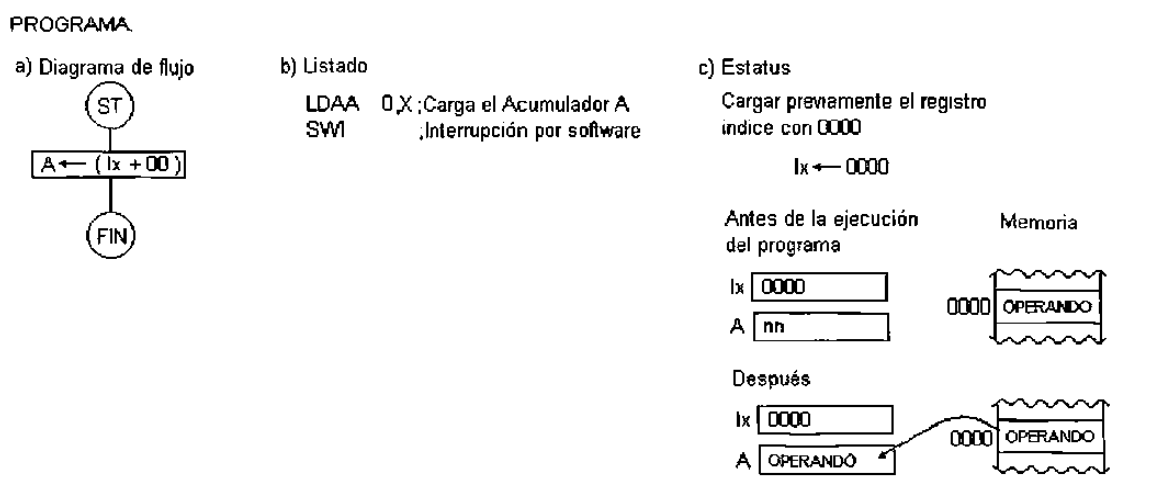


Fig. 3.14 Programa que carga al acumulador A en modo indexado X

EJEMPLO.- 11

Usando el modo indexado depositar el contenido del acumulador A en la dirección 0001

Instrucción - almacenar el acumulador A en modo indexado

Código de instrucción -A7

Mnemónico - STAA

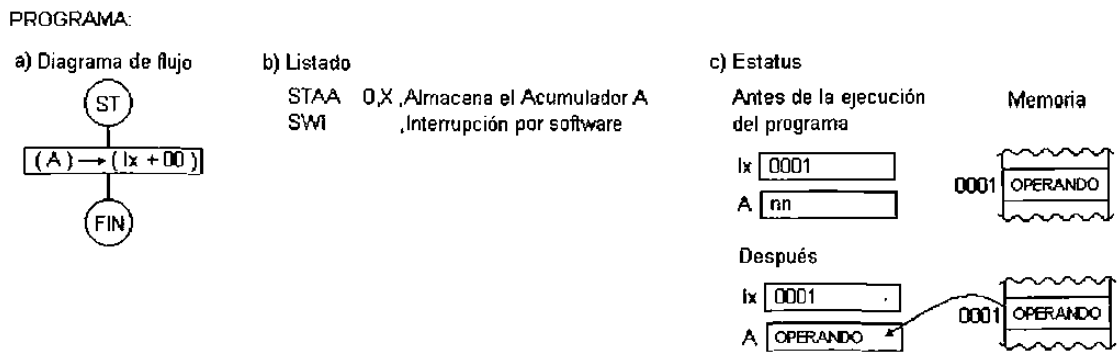


Fig. 3.15 Programa que almacena el acumulador A en modo indexado X

3.6 MODO RELATIVO

En este modo la instrucción está formada por 2 bytes, uno para el código de la instrucción y el siguiente para un número de 8 bits que indica la cantidad de lugares de memoria que el MPU tiene que avanzar o retroceder para encontrar el operando, a este número también se le llama offset, si es positivo el MPU avanza y si es negativo retrocede.

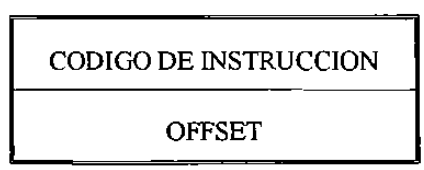


Fig 3.16 Estructura de una instrucción del modo relativo

Las instrucciones en modo relativo se clasifican en condicionadas y no condicionadas, el concepto de condicionamiento significa que para que se ejecute la instrucción, es necesario que se presente en el registro de código de condición el estatus para el cual fue diseñada la instrucción, a este tipo de instrucciones se les llama branch, ramificaciones o bifurcaciones condicionadas, cuando la instrucción no interroga o prueba el estado del registro de código de condición es una instrucción no condicionada.

Con respecto al cálculo de offset la mayoría de los sistemas de apoyo (software de apoyo) tienen una rutina para efectuarlo, si se desea hacer manualmente es necesario calcular la diferencia entre la dirección final y la dirección inicial y recordar que el contador del programa se encuentra un lugar adelante del que se está ejecutando o leyendo, por tal motivo la fórmula para calcular el offset es la sig.: offset = dirección final - (dirección inicial + 2).

EJEMPLO.- 12

Usar la instrucción de branch no condicionado - branch always.

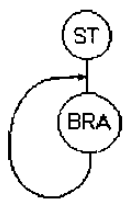
Instrucción - branch always

Código de instrucción -20

Mnemónico - BRA

PROGRAMA:

a) Diagrama de flujo



b) Listado

```
Lazo    BRA   Lazo  ;Brinco incondicionado
                    ;a la misma instrucción
```

c) Estatus

Al ejecutar el programa, éste se ciclará brincando a la misma dirección de la instrucción

Fig 3.17 Programa ejemplo para verificar el adecuado cálculo del Offset.

Para usar una instrucción de tipo condicionada es necesario haber efectuado una instrucción previa que altere el contenido del registro de código de condición, de tal manera que el estado resultante de dicha operación quede indicada por los bits de este registro y estonces así usar la instrucción condicionada.

EJEMPLO.- 13

Probar la instrucción branch if carry set

Instrucción - ramifica si el bit de acarreo es 1

Código de instrucción -25

Mnemónico - BCS

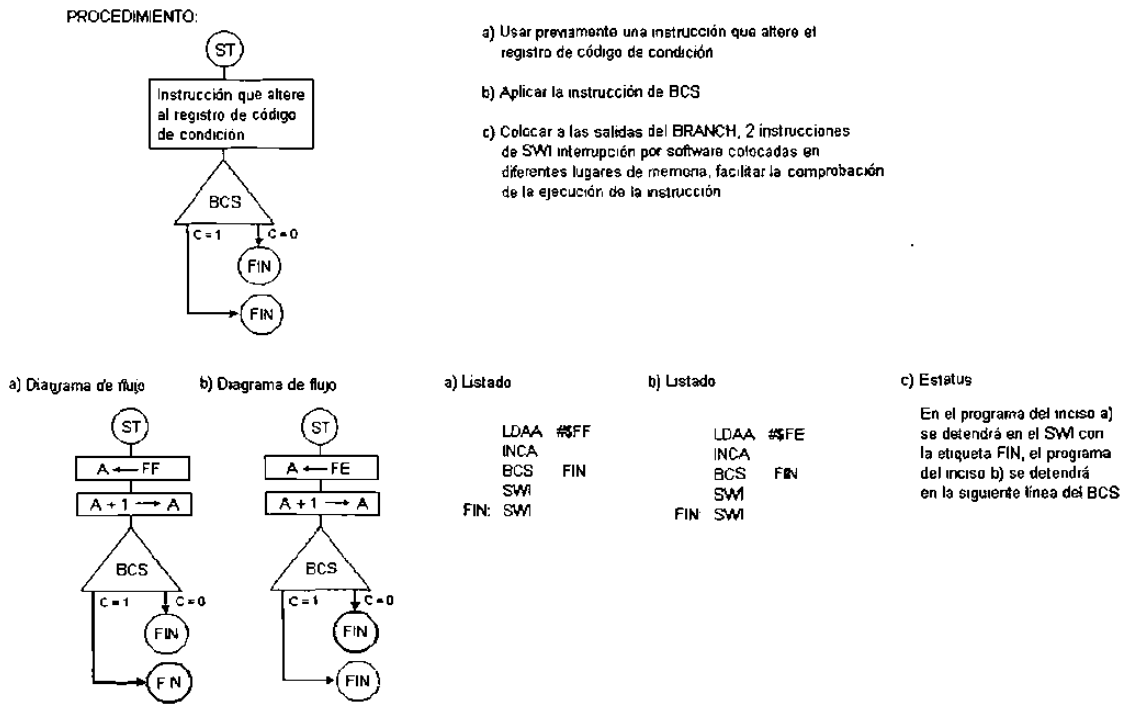


Fig. 3.18 Programa de prueba del modo relativo

PROGRAMAS PROPUESTOS

1.- Usando los modos de direccionamiento directo, extendido e indexado, diseñar un programa para cada caso que transfiera el contenido de la localidad 50 a la 51.

2.- Diseñar un programa para cada caso, que efectua la prueba de las siguientes instrucciones en modo inherente: ABA, CLRA, CBA, COMA, DECA, INCA, LSRA, TAB.

3.- Diseñar un programa para cada caso que efectue la prueba de las siguientes instrucciones en modo relativo: BCC, BEQ, BGT, BNE, BPL.

CAPITULO 4

Estrategias básicas para el Diseño de Programas (Práctica No. 3)

OBJETIVO

El objetivo de este tema es el de apoyar el proceso de aprendizaje de la programación con Microprocesadores revisando una serie de modelos de programación que consisten en técncas básicas de manipulación de datos usando los recursos que ofrece el microprocesador y que pueden dar solución a un conjunto de problemas semejantes.

4.1 PROGRAMAS ITERATIVOS O DE LAZO

Un programa iterativo o de lazo hace que el Microprocesador efectúe en forma repetitiva una secuencia de instrucciones. Los programas de lazo tienen cuando menos las siguientes cuatro secciones:

1 - SECCION DE INICIO, en la que se establecen los valores iniciales de variables, contadores, indicadores o registros de dirección, etc.

2 - SECCION DE PROCESO, donde se efectúa la manipulación de los datos, esta sección es la que efectúa el trabajo o tarea de lazo.

3 - SECCION DE CONTROL DE LAZO, la que actualiza los contadores e indicadores para efectuar la siguiente iteración.

4 - SECCION DE CONCLUSION, la que analiza y almacena los resultados del programa.

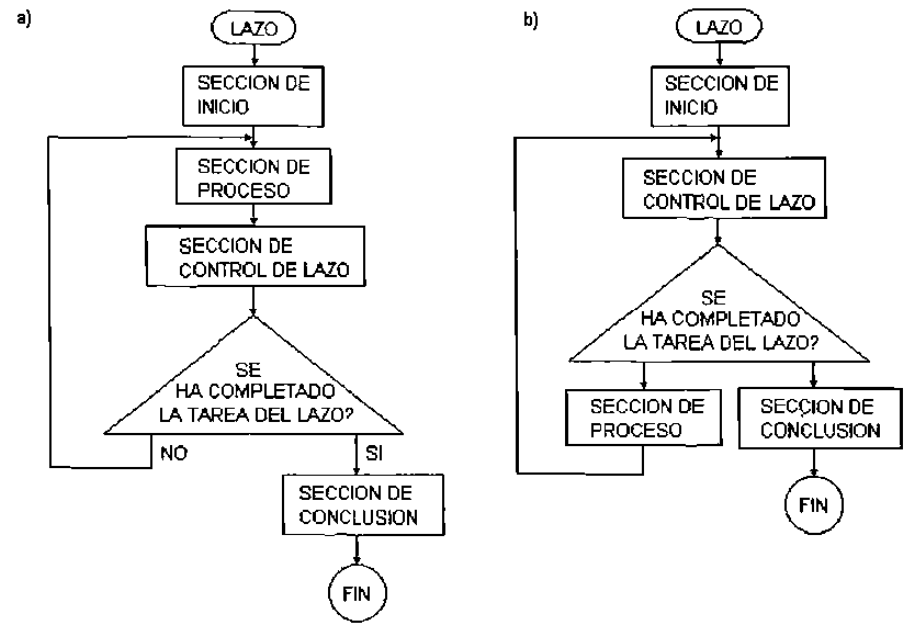


Fig 4.1 Diagrama de flujo de programas de lazo, a) lazo típico, b) lazo que permite cero iteraciones.

EJEMPLO.- 4.1

Tomar el contenido de la localidad 0050 y contar el número de bits igual a 1 que contenga, colocar el resultado en la dirección 0051.

Solución:

Dado que la cantidad total de bits que contiene una palabra contenida en la localidad 0050 es de 8, entonces el número de iteraciones que tendrá que verificar el lazo es también de 8, por tal motivo habrá que establecer un contador del número de iteraciones al inicio del programa, que se decremente en uno a cada iteración, la dirección 51 puede operar como un contador de número de bits igual a uno, para detectar si los bits son igual a uno se puede efectuar un corrimiento a la derecha, haciendo que el dato se cargue en el bit de acarreo del registro de código de condición y posteriormente hacer una pregunta mediante un branch si el bit de acarreo es igual a uno.

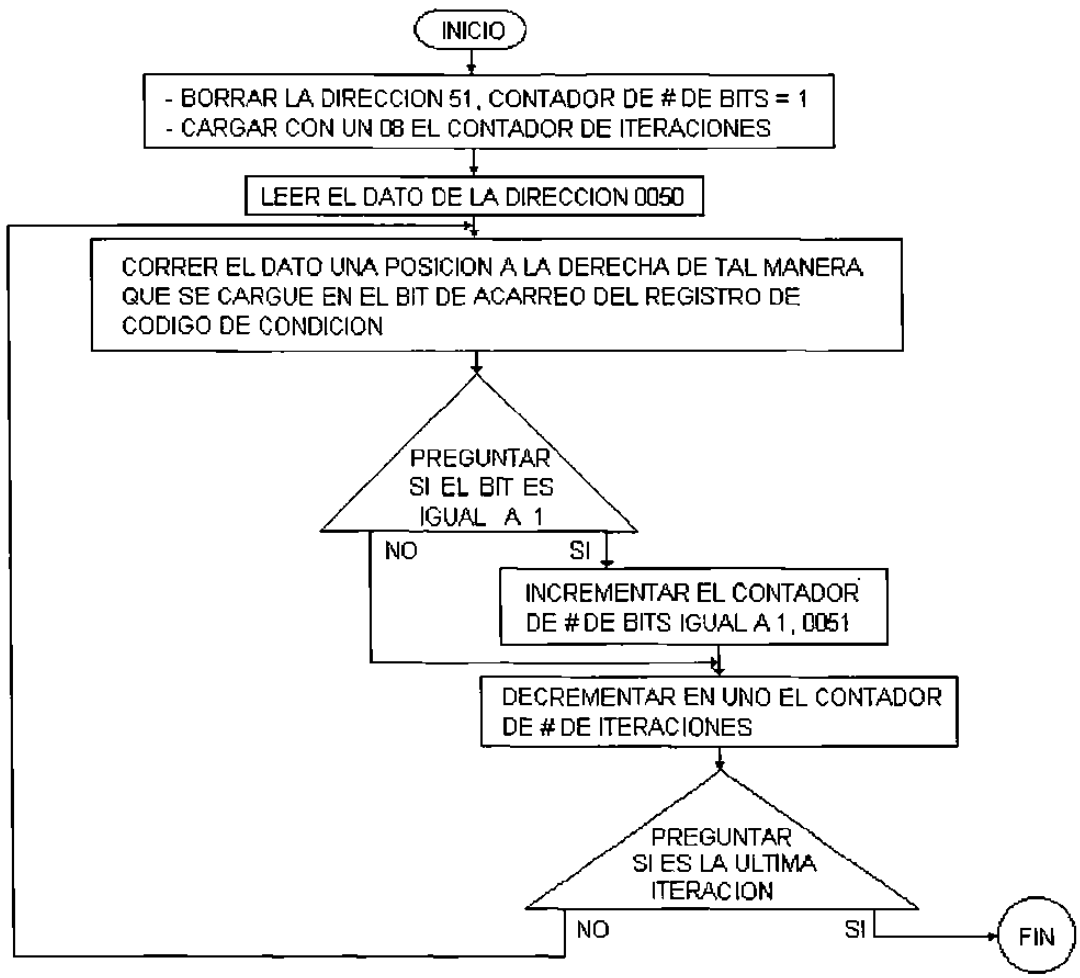
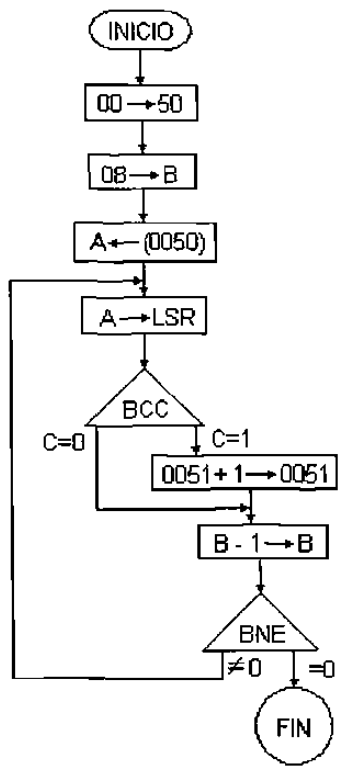


Fig. 4.2 Diagrama de flujo de un programa que cuenta los bit con valor 1 de la dirección 0050

b) Diagrama de flujo de detalle



c) Listado

```
        CLR   $0051
        LDAB  #$08
        LDAA  $0050
LAZO1;  LSRA
        BCC   LAZO
        INC   $0051
LAZO;   INCB
        BNE   LAZO1
        SWI
```

d) Estrategia de prueba

1- CARGAR EN LA DIRECCION 50
UN DATO CONOCIDO, ES DECIR
DE BITS =1 PREVIAMENTE
CONOCIDO POR EJEMPLO:

DATO	# BITS =1
00	0
01	1
03	2
07	3
0F	4
1F	5
3F	6
7F	7
FF	8

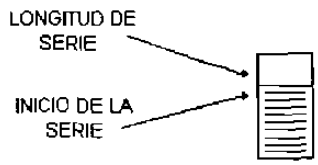
2- CORRER EL PROGRAMA

3- VERIFICAR EL RESULTADO EN LA
DIRECCION 0051-

Fig. 4.3 Programa que cuenta los bits igual a 1 de la 0050

4.2 SERIES DE NUMEROS

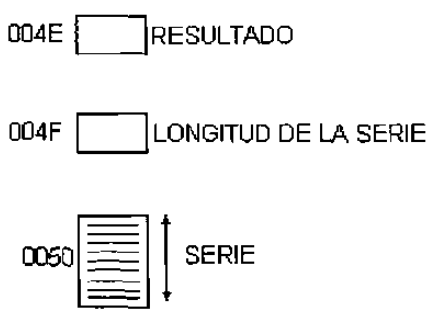
Una serie de números es un arreglo continuo de datos numéricos de los que se conoce generalmente la dirección de inicio de la serie y su longitud.



Para manejar una serie de números se recomienda usar como registro indicador o apuntador al registro indice y usar como contador de número de iteraciones el dato de la longitud de la tabla.

EJEMPLO.- 4.2

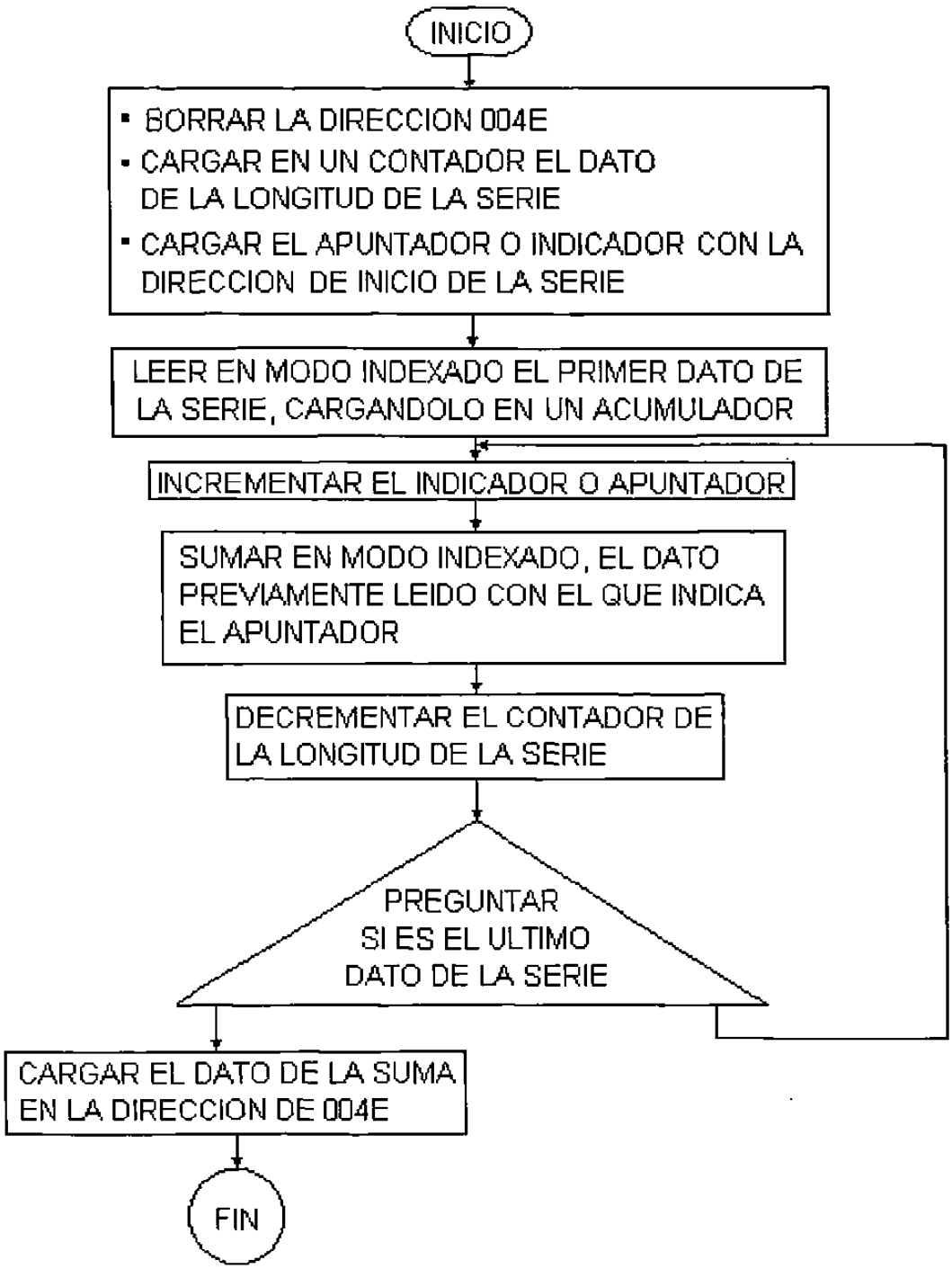
Efectuar la suma sin acarreo de una serie de números que inicia en la dirección 0050 y cuya longitud se indica en la dirección 004F, colocar el resultado de la suma en la dirección 004E.



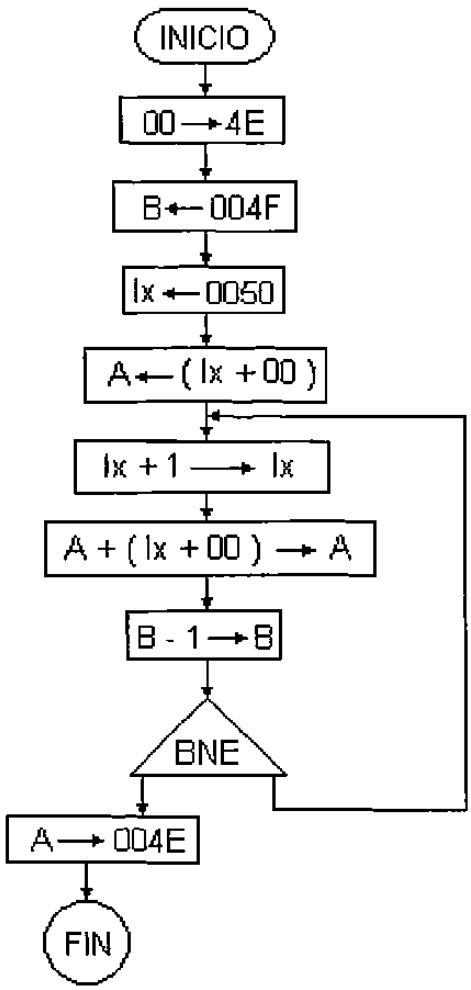
Solución:

Tomar el dato de la longitud de la serie como el contador de iteraciones, cargar el registro apuntador o indicador con la dirección de inicio de la serie, leer el pirmer dato de la serie en modo indexado, incrementar el indicador y sumar con el siguiente, así sucesivamente tantas veces como lo permita el contador de iteraciones, por último colocar el resultado en la dirección 004F.

Diagrama de flujo de la idea



b) Diagrama de flujo de detalle



c) Listado

```
        CLR  $4E      ;Borra memoria resultado
        LDAB $4F      ;carga contador en B
        LDX  #$004F;incializa apuntador
        LDAA 0,X      ;carga primer operando
LAZO:   INX           ;incrementa apuntador
        ADDA 0,X      ;suma operando
        DECB          ;decrementa longitud
        BNE  LAZO     ;último operando?
        STAA $4E      ;deposita resultado
        SWI           ;interrupcion por software
```

d) Estrategias de prueba

1 - Cargar la serie con #'s conocidos.
2 - Cargar la longitud de la serie.
3 - Correr el programa.
4 - Verificar el resultado leyendo la dirección 004E.

4.3 LAZOS DE RETARDO DE TIEMPO

Un lazo de retardo es un programa que efectúa como objetivo, la generación de un intervalo o retardo de tiempo, y son ampliamente utilizados como parte fundamental de los programas de entrada/salida.

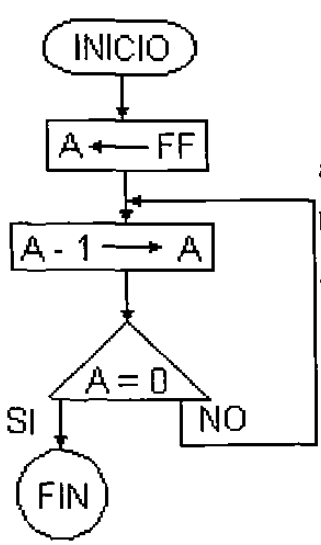
El retardo de tiempo o intervalo se logra aprovechando el tiempo que transcurre durante la ejecución de una instrucción, ejecutandola en forma repetitiva, tantas veces como sea necesario hasta obtener el retardo deseado.

El tiempo de ejecución de una instrucción, se obtiene, multiplicando el número de ciclos de máquina empleados para su ejecución por la duración del ciclo de máquina, suponiendo que cada ciclo de máquina dura un microsegundo

t_{cm} = Número de ciclos X 1microseg.

EJEMPLO.- 4.3

Programa de retardo usando un acumulador como contador decrementandolo continuamente.



Tiempo = [a + (b +c)*256 +d]ciclos x t_{cm}

Donde:

a=2 ciclos máquina de la instrucción LDAA en modo inmediato

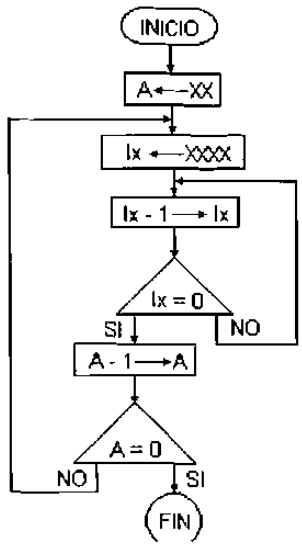
b=2 ciclos máquina de la instrucción DECA en modo inherente

c=3 ciclos máquina de la instrucción BNE en modo relativo

d=14 ciclos máquina de la instrucción SWI en modo inherente

EJEMPLO.- 4.4

Programa de retardo usando un acumulador y el registro indice X



$T = \{a+[b+(c+d)*(xxxx)+e+f)*(xx)+g]ciclos*t_{cm}$

donde:

a=2 Ciclos máquina de la instrucción LDAA en modo inmediato

b=3 Ciclos máquina de la instrucción LDX en modo inmediato

c=3 Ciclos máquina de la instrucción DEX en modo inherente

d=3 Ciclos máquina de la instrucción BNE en modo relativo

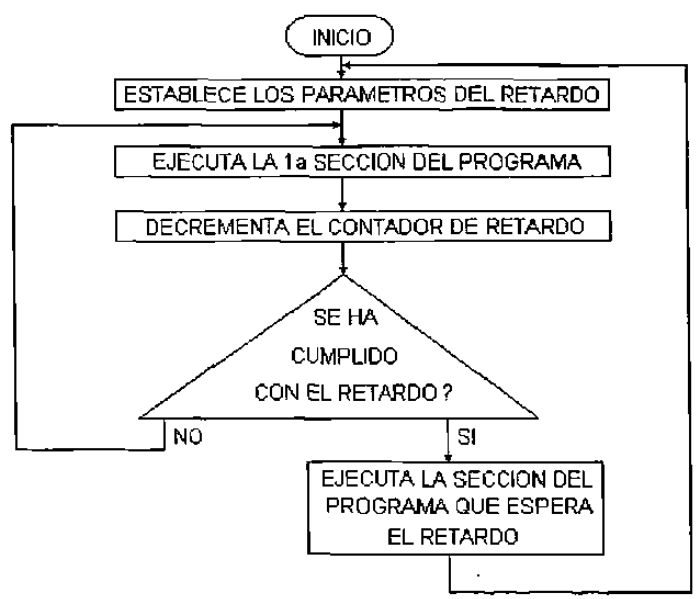
e=2 Ciclos máquina de la instrucción DECA en modo inherente

f=3 Ciclos máquina de la instrucción BNE en modo relativo

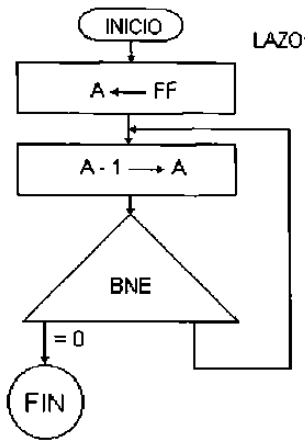
g=14 Ciclos máquina de la instrucción SWI en modo inherente

Los programas de retardo anteriormente descritos hacen que el MPU atienda solamente al programa de retardo, sin embargo cuando el programa de retardo interactúa con o es parte de algun otro programa, se hace necesario entonces que el MPU atienda a ambos, convirtiendo al programa de retardo de pasivo a activo.

EJEMPLO.- 4.5



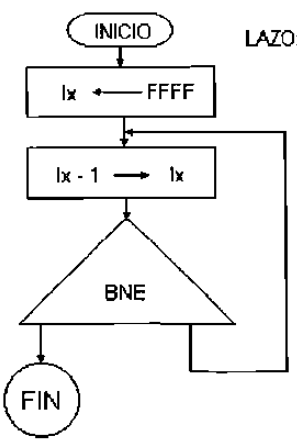
b) Diagrama de flujo en detalle



c) Listado

```
        LDAA  #$FF  ;establece retardo
LAZO:   DECA        ;decrementa lazo
        BNE   LAZO  ;fin de lazo?
        SWI         ;interrupción por software
```

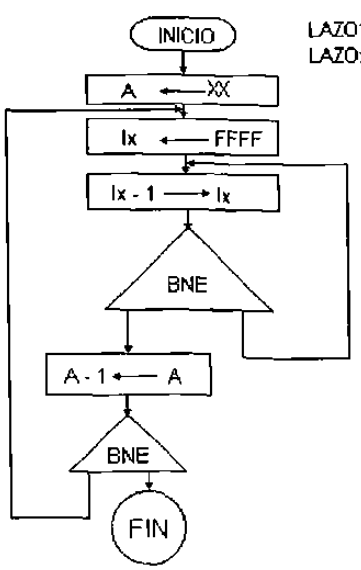
b) Diagrama de flujo en detalle



c) Listado

```
        LDX   #$FFFF ;establece retardo
        DEX          ;decrementa lazo
LAZO:   BNE   LAZO   ;fin de lazo?
        SWI          ;interrupción por software
```

b) Diagrama de flujo en detalle



c) Listado

```
        LDAA  #$XX   ;establece xx retardos
LAZO1:  LDX   #$FFFF ;establece retardo base
LAZO:   DEX          ;decrementa lazo?
        BNE   LAZO   ;fin de lazo
        DECA         ;decrementa lazo xx
        BNE   LAZO1  ;fin de lazo?
        SWI          ;interrupción por software
```

Estrategias de prueba

a) Para todos los casos

Para correr el programa, se notara que durante un intervalo de tiempo el MPU atiende la ejecución del programa de retardo, al concluir retorna a atender al usuario.

4.4 TABLAS DE DATOS

Una tabla es una estructura de datos básica, por lo regular continua, de la cual se conoce sus direcciones inicial y final y cuyos datos que la componen por lo general tienen la misma característica es decir la información o el parámerto que representan es similar para todos los datos.

4.4.1 Solución de Algoritmos Númericos mediante Tabla de Datos

La principal aplicación de las tablas es como tabulaciones de búsqueda o look up tables que consisten en tabulaciones absolutas de datos y que son ampliamente utilizadas como opción a los algoritmos numéricos que tradicionalmente dan solución a problemas como: conversiones de código, operaciones aritméticas complejas, linearización, interpretación de variables de entrada, respuesta a comandos o instrucciones, acceso a archivos, etc.

Una tabla es un método simple de resolver problemas o tomar decisiones, basta con organizarla adecuadamente para que una variable de entrada o independiente, encuentre fácilmente la variable dependiente, de salida o equivalente.

DIRECCION		CONTENIDO
VARIABLE INDEPENDIENTE LLAVE O CLAVE DE ENTRADA	-	VARIABLE DEPENDIENTE RESULTADO O SALIDA
A X B	=	RESULTADO DE LA MULTIPLICACION
A	=	Log A
A	=	A^2
CODIGO A	=	CODIGO B
BINARIO	=	DECIMAL
BINARIO	=	GRAY
BCD	=	7 SEGMENTOS

Tabla 4.1

Una vez cargada la tabla en memoria, su acceso se efectúa manipulando un acumulador en modo indexado. En el ejemplo 4.7 se ilustra un un programa para convertir de código binario a código BCD usando una tabla.

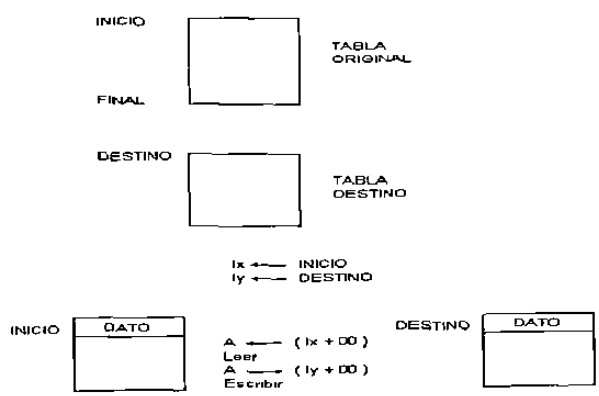
El tiempo de acceso a la tabla se reduce a cargar un acumulador en modo indexado.A diferencia de un algoritmo o ecuación que requiere efectuar una serie de instrucciones, cálculos, funciones lógicas, iteraciones, etc.

Una desventaja de las tablas es que ocupan un gran espacio en memoria, sin embargo en la actualidad la memoria es económica y compacta.

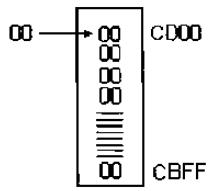
4.4.2 Manipulación de Datos mediante una Tabla

Cuando se cuenta con una tabla de datos variabes, por lo general surge la necesidad de obtener información de ésta, tal podria ser calcular el promedio de los valores, encontrar el número mayor, ordenar los valores con algun criterio, etc. Estas operaciones implican efectuar la misma operación a todos y cada uno de los datos de la tabla, por lo que aparece la necesidad de construir un algoritmo tipo lazo iterativo.

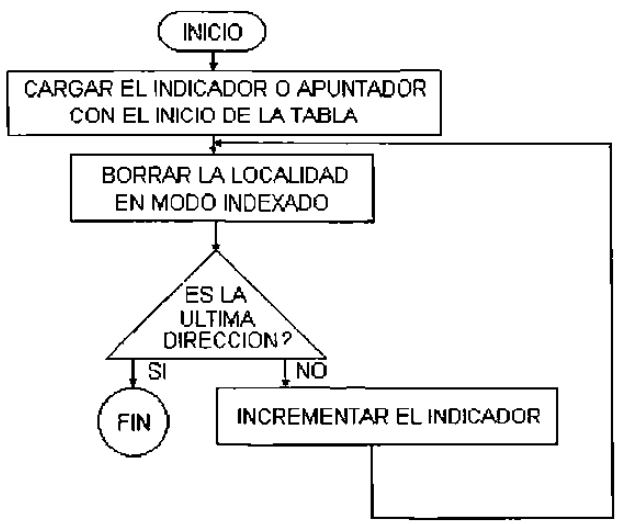
Con respecto a la manipulación de los datos en una tabla, a menudo es necesario usar dos indicadores o apuntadores para este caso es muy útil contar con dos registros índice, tal es el caso de mover un dato de una tabla a otra, se puede usar un registro para leer el dato y el otro registro indice para escribir el dato en la otra tabla.



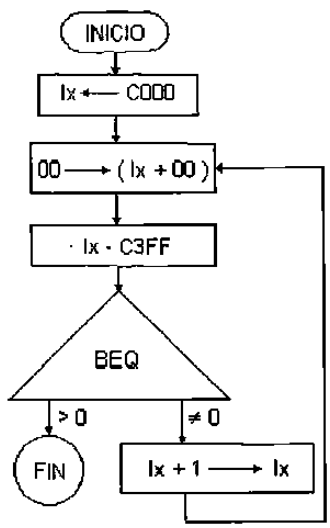
Ejemplo 4.6 Diseñar un programa que borre (escriba ceros) en una tabla.



a) Diagrama de flujo de la idea



b) Diagrama de flujo de detalle



c) Listado

```
        LDX   #$C000
LAZO1;  CLR   0,X
        CPX   #$C3FF
        BEQ   LAZO
        INX
        BRA   LAZO1
LAZO;   SWI
```

d) Estrategias de prueba

1.- Verificar previamente el contenido de las direcciones a borrar.
2.- Correr el programa.
3.- Verificar que la tabla ha sido borrada.

Ejemplo 4.7 Diseñar un programa que por medio de una tabla efectúe una conversión de código binario a BCD.

Procedimiento: El dato binario puede cargarse en el registro indice y posteriormente leer en modo indexado el dato de la tabla que a su vez ha sido cargada previamente con los equivalentes en BCD, para muestra basta con una tabla de 16 datos.

```
0000 - 7F - CLR EXT
   1 - 00
   2 - 4E
   3 - D6 - LDA B DIR
   4 - 4F
   5 - CE - LDX
   6 - 00
   7 - 50
   8 - A6 - LDA A X
   9 - 00
   A - 08 - INX INH
   B - AB - ADD A X
   C - 00
   D - 5A - DEC B INH
   E - 26 - BNE REL
   F - □
  10 - 97 - STA A DIR
  11 - 4E
  12 - 3F - SWI
```

a apartir de la dirección C000

```
C000 - 00
   1 - 01
   2 - 02
   3 - 03
   4 - 04
   5 - 05
   6 - 06
   7 - 07
   8 - 08
   9 - 09
   A - 10
   B - 11
   C - 12
   D - 13
   E - 14
   F - 15
```

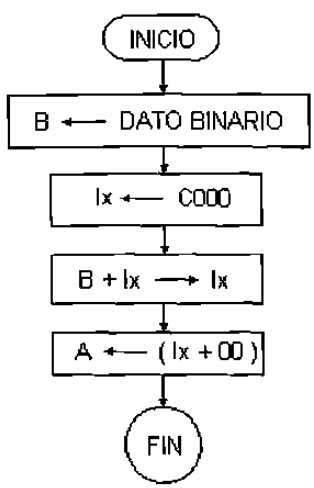
2 - El dato en binario se carga en el acumulador B.

3 - El Ix se carga con la dirección de inicio de la tabla.

Ix ------- C000

4 - X se usa la instrucción ABX suma el acumulador B al Ix para formar la dirección de dato binario.

5 - Por último lee por medio del acumulador A el dato de la tabla.



Listado

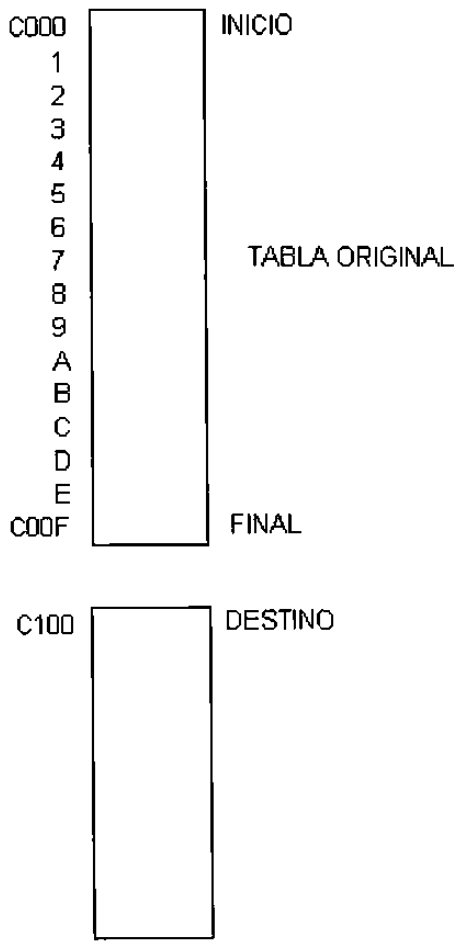
```
LDAB    #DATO
LDX     #$C000
ABX
LDAA    0,X
SWI
```

Procedimiento de prueba

1 - Cargar la tabla.
2 - Cargar en la dirección 0001 el dato binario que se desee convertir a decimal.
3 - Correr el programa.
4 - El dato decimal estará en el acumulador A.

Ejemplo 4.8 Diseñar un programa que transfiera una tabla.

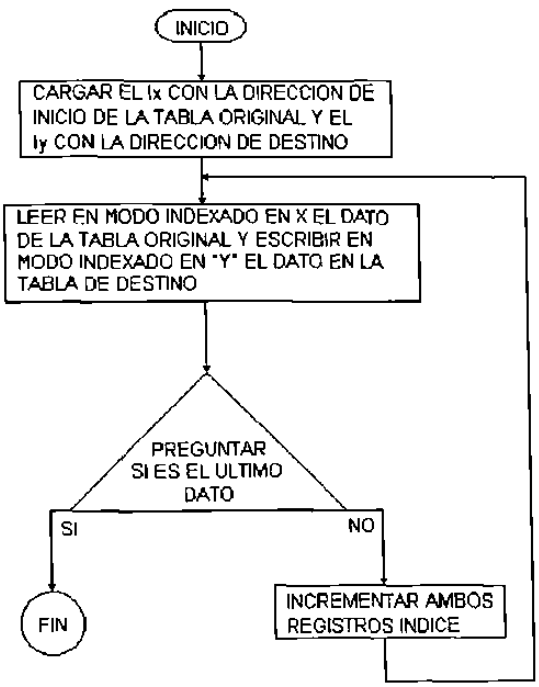
Procedimiento: Establecer la tabla original apartir de la dirección C000, para este caso de prueba usar una tabla pequeña por ejemplo de 16 datos, por último establecer el destino de la tabla en la dirección C100.



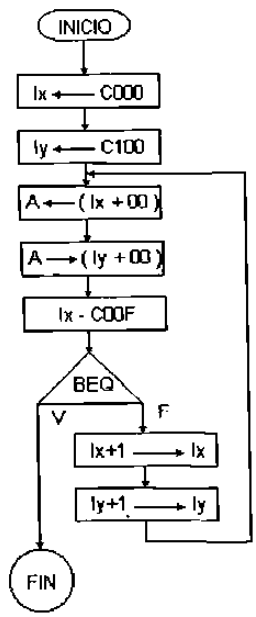
1 - Usar el Ix para leer el dato de la tabla origina

2 - Usar el Iy para escribir la tabla transferida.

a) Diagrama de flujo de la idea



b) Diagrama de flujo en detalle



c) Listado

```
        LDX   #$C000
        LDY   #$C100
LAZO;   LDAA  0,X
        STAA  0,X
        CPX   #$C00F
        BEQ   LAZO
        INX
        INY
        BRA   LAZO1
LAZO;   SWI
```

PROGRAMAS PROPUESTOS

1 - De una serie de números que inicia en la dirección 0050 y cuya longitud se indica en la dirección 004F, contar la cantidad de números positivos que contiene la serie, y colocar el resultado en la dirección 004E.

2 - De una serie de números que inicia en la dirección 0050 y cuya longitud se indica en la dirección 004F, contar la cantidad de números positivos, negativos y cero, contenidos en la serie, y colocar los resultados en las direcciones, 004C - positivos, 004D - negativos y 004E - cero.

3 - De una serie de números que inicia en la dirección 0050 y cuya longitud se indica en la dirección 004F, separar el número mayor y colocarlo en la dirección 004E.

4 - De una tabla que inicia en la dirección C000 y finaliza en la dirección C00F, ordenar las cantidades contenidas en ella en forma ascendente.

4.5 SUBRUTINAS

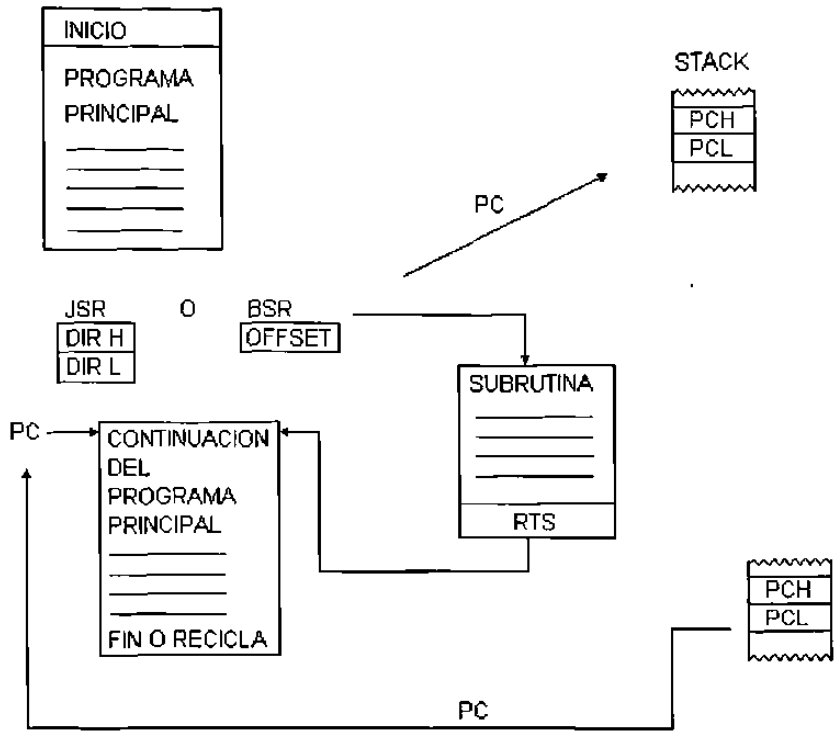
Una subrutina es un programa que cumple con alguna función específica. Las subrutinas pueden accesarce o ser llamadas por el programa principal una o tantas veces como sea necesario. La ventaja inmediata del uso de una subrutina que se emplea varias veces, es que no es necesario integrarla al programa principal, logrando así su simplificación y reducción de tamaño.

Las instrucciones asociadas a las subrutinas son, *JSR, JUMP TO SUBROUTINE* (brinco a subrutina) en modo extendido e indexado, *BSR, BRANCH TO SUBROUTINE* (ramificación a subrutina) en modo relativo y *RTS, RETURN FROM SUBROUTINE* (retorno de subrutina) en modo inherente.

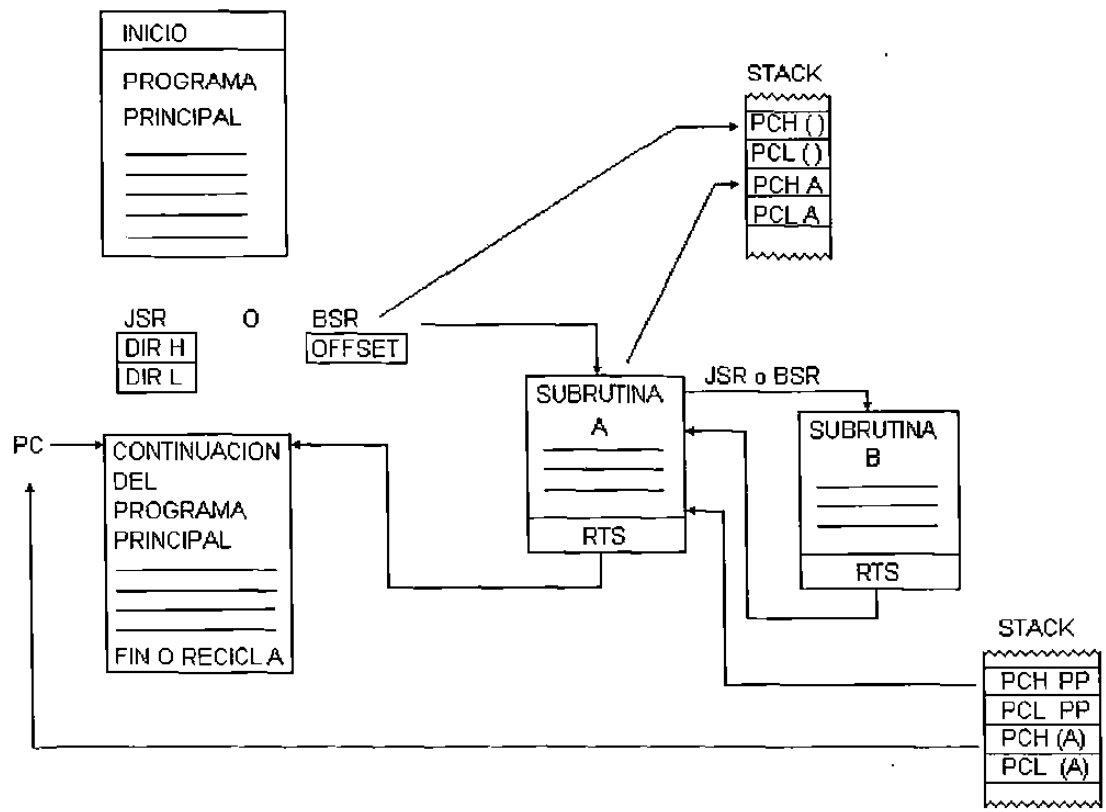
Cuando el programa principal llama a una subrutina cede el control del programa a la subrutina hasta realizar su ejecución y posteriormente retorna el control al programa principal.

Al ejecutarse las instrucciones de *JSR* o *BSR* el contenido del contador del programa, que indica la siguiente instrucción a ejecutar del programa principal, se almacena temporalmente en el STACK, posteriormente se carga el contador del programa con la dirección de la primer instrucción de la subrutina e inicia así su ejecución.

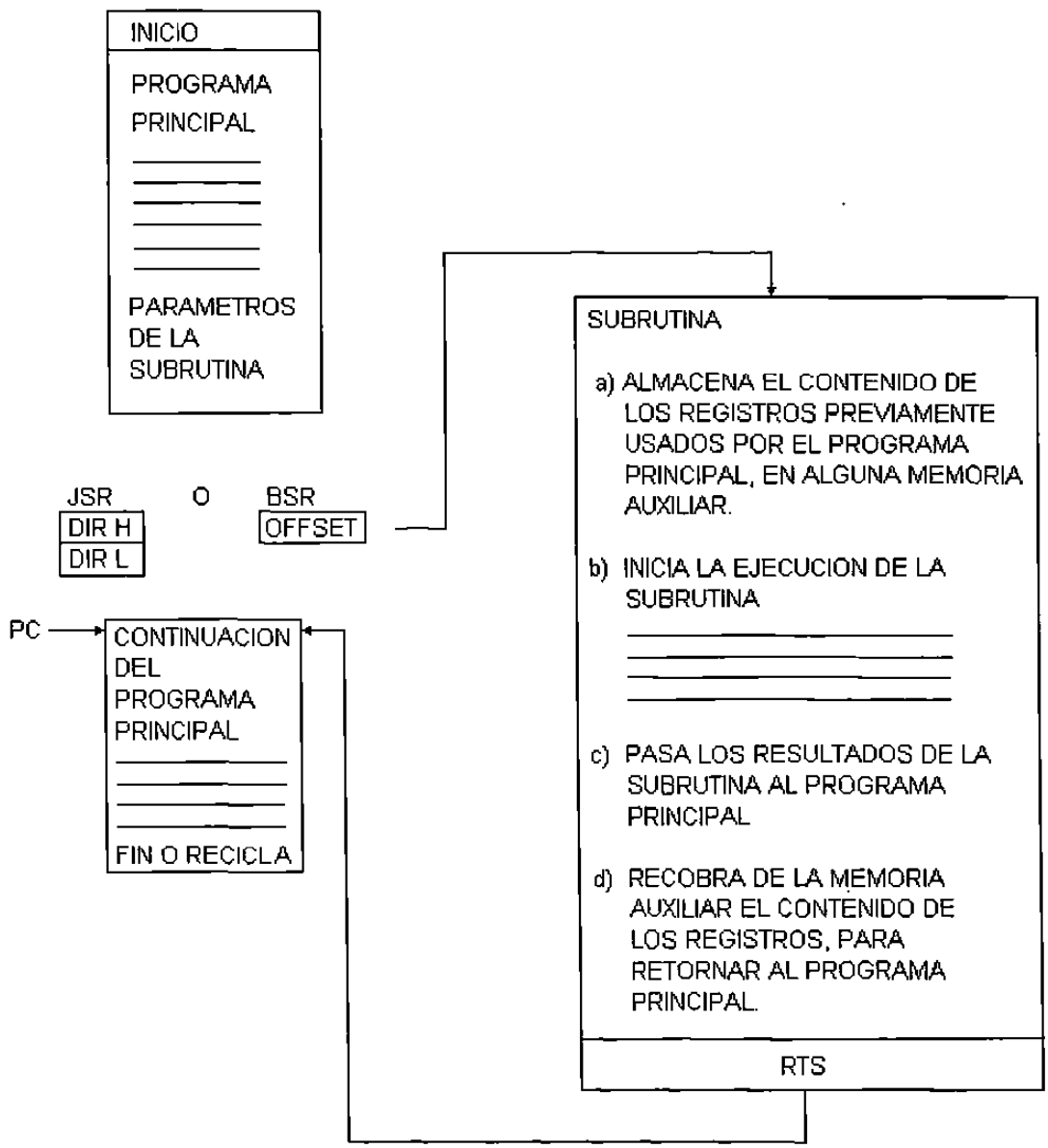
Al encontrar el MPU una instrucción de *RTS*, durante la ejecución de la subrutina, recobra el contenido del contador del programa que fuera anteriormente almacenando en el STACK para continuar con la ejecución del programa principal.



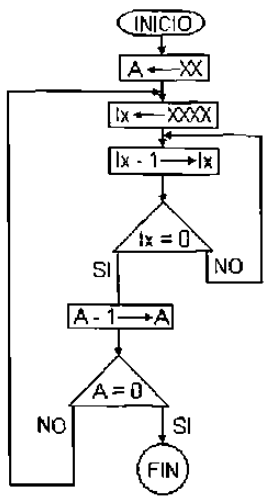
Este proceso puede anidarse, es decir una subrutina puede a su vez llamar a otra subrutina y así sucesivamente, tantas veces como lo permita el tamaño del STACK que almacena el contador del programa para retornar en cada uno de los casos.

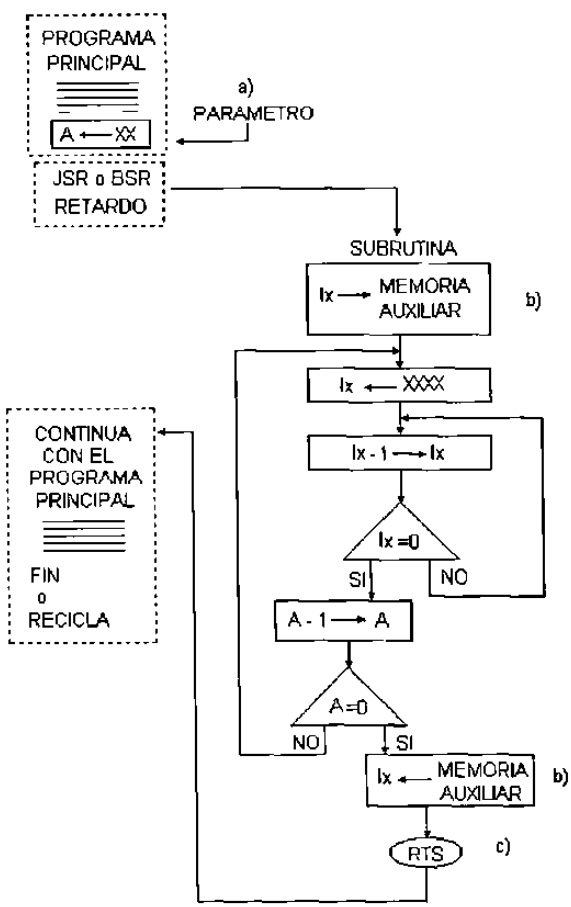


Al diseñar una subrutina, es necesario considerar que pudiera usar para su ejecución los mismos registros que usa el programa principal, es también recomendable considerar que las subrutinas requieren de información previa, llamada también parámetros de la subrutina, y que la subrutina generará resultados como consecuencia de su ejecución que tendrán que ser usados por el programa principal. La siguiente es una estructura típica para una subrutina.



Ejemplo 4.9 Convertir en una subrutina al programa de retardo de tiempo que una el acumulador A y el registro indice X, considerar como parámetro de la subrutina el valor del acumulador A que determina el número de veces que se ejecutará la sección de retardo del Ix.





COMENTARIOS

a) PARAMETRO, CARGAR EL ACUMULADOR A CON EL # DE VECES A EJECUTAR LA RUTINA ANTES DE LLAMARLA

b) EN LA SECCION DE LA SUBRUTINA ALMACENAR EL Ix ES UNA MEMORIA AUXILIAR Y POSTERIORMENTE RECOBRARLO ANTES DE REGRESAR AL PROGRAMA PRINCIPAL

c) LA INSTRUCCION DE SWI SE CAMBIA POR UN RTS

PROGRAMAS PROPUESTOS

1 - Convertir en subrutinas los programas diseñados en los temas anteriores.

CAPITULO 5

Puerto Paralelo (Práctica No. 4)

OBJETIVO

El objetivo de esta práctica es de que el alumno se familiarice con el uso de los diferentes puertos paralelo con los que cuenta el microcontrolador MC68HC11E2.

INTRODUCCION

El microcontrolador MC68HC811E2 cuenta con 5 puertos de propósito general (E/S) totalizando 15 líneas programables de entrada/salida, 12 líneas de salida y 11 de entrada.

En la tabla 5.1 se muestra la operación y la dirección del registro de datos y el registro de direccionamiento de datos en el caso de los puertos de (entrada/salida).

PUERTO DE SALIDA

Para la prueba de esta operación se seleccionó al Puerto B (PB) para lo cual se implementó un grupo de ocho LED's cuya corriente es manejada con un inversor (4049) conectado como se muestra la Fig. 1

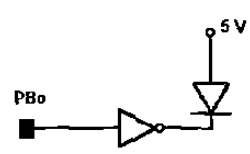


Fig. 5.1 Conexión de un LED para controlarse por el Puerto B

PUERTO	OPERACION									DIRECCION (HEX)
		PA7	PA6	PA5	PA4	PA3	PA2	PA1	PA0	
A		E/S	S	S	S	S	E	E	E	REGISTRO DE DATOS 1000
		PB7	PB6	PB5	PB4	PB3	PB2	PB1	PB0	
B		S	S	S	S	S	S	S	S	REGISTRO DE DATOS PERIFERICOS 1004
		PC7	PC6	PC5	PC4	PC3	PC2	PC1	PC0	
C		E/S	E/S	E/S	E/S	E/S	E/S	E/S	E/S	REGISTRO DE DATOS PERIFERICOS 1003
		DRC7	DRC6	DRC5	DRC4	DRC3	DRC2	DRC1	DRC0	REGISTRO DE DIRECCIONAMIENTO DE DATOS 1007
		PD7	PD6	PD5	PD4	PD3	PD2	PD1	PD0	
D		0	0	E/S	E/S	E/S	E/S	E/S	E/S	REGISTRO DE DATOS PERIFERICOS 1008
		0	0	DRD5	DRD4	DRD3	DRD2	DRD1	DRD0	REGISTRO DE DATOS PERIFERICOS 1009
		PE7	PE6	PE5	PE4	PE3	PE2	PE1	PE0	
E		E	E	E	E	E	E	E	E	REGISTRO DE DATOS PERIFERICOS 100A

Tabla 5.1 Puertos de entrada/salida del microprocesador MC68HC11E2

De acuerdo con la conexión, la lógica de operación de LED se indica en la Tabla 5.2

PBo	LED
0	APAGADO
1	ENCENDIDO

Tabla 5.2

PRUEBA PRELIMINAR

Debido a que el Puerto B es de solo salida, para el control de éste solo se tiene el registro de datos periféricos ($ 1004).

Dentro del sistema operativo PCBUG11con el comando MODIFY MEMORY (MM) Despliegue la dirección $1004. Escriba cualquier dato, observe que la combinación del LED encendido y apagados corresponde al dato que se escriba en el registro de datos.

PRUEBA A TRAVES DE UN PROGRAMA

Efectúe la msma prueba a través de un programa . El programa consistirá solamente en transferir un dato, ya sea de un acumulador o de una localidad de memoria al registro de datos del puerto B.

PUERTO DE ENTRADA

Para la operación de entrada se utilizará el puerto C, al cual se le conectará un minidip con 8 interruptores como se muestra en la siguiente figura.

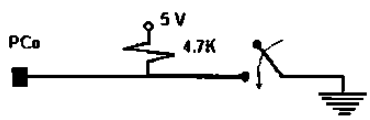


Fig. 5.2 Conexión de un interruptor al
Puerto de Entrada C

La lógica de operación se indica en la siguiente tabla.

INTERRUPTOR	PCo
ABIERTO	1
CERRADO	0

Tabla 5.3

PRUEBA PRELIMINAR

Es necesario previamente programar al puerto como entrada ya que el Puerto C puede operar tanto entrada como salida.

Con el comando MM escriba 00 en el registro de direccionamiento de datos del Puerto C, coloque cualquier combinación de interruptores y despliegue el registro de datos.

Observe que la combinación de unos y ceros que se configuró en los mini-interruptores es la misma que contiene el registro de datos del Puerto C $1003. Si se cambia de nuevo la información en la entrada del puerto se tendrá que desplegar también el registro de datos, una opción rápida es avanzar y retroceder con las flechas verticales.

PRUEBA A TRAVES DE UN PROGRAMA

Elabore un programa para que los LED's conectados al Puerto B sean controlados por los interruptores del Puerto C.

PROGRAMA EJEMPLO

Diseñe un programa que genera un contador binario natural controlando su paro y arranque con el PC0 con intervalos de 0.5 seg entre incrementos.

PROGRAMAS PROPUESTOS

1.- Diseñe un programa que simule los siguientes operadores, mostrados en la figura 5.3

Fig 5.3 Operadores lógicos para simularse en el programa 1

2.- Diseñe un programa que simule un decodificador de línea de 3 x 8, con salidas activas a nivel bajo.

3.- Diseñe un programa que genere un contador binario natural que incremente con la transición negativa del Puerto C.

4.- Diseñe un programa que saque por el Puerto C, los datos de una tabla cuyo inicio se indica en la dirección 00D0 y 00D1 y la longitud en la dirección 00D2, con un intervalo de tiempo entre dato y dato de 5 min.

CAPITULO 6

Puerto de Comunicación Serie Asíncrono (SCI) (Práctica No. 5)

OBJETIVO

El objetivo de esta práctica es de que el alumno conozca el uso del puerto serie asíncrono (SCI - Serial Communication Interfase) del MC68HC11E2.

INTRODUCCION

El microcontrolador MC68HC811E2 cuenta con dos sub-sistemas independientes para la comunicación serie, uno de ellos esta concebido para la comunicación serie con periférico (SPI), desarrollándose en este caso una comunicación serie síncrona.

Mientras el segundo subsistema (SCI) consta de un Receptor/Transmisor Universal Asíncrono (UART) para interconectar el MCU con una terminal, una PC o cualquier otro dispositivo que cuente con un Puerto Serie Asíncrono.

El SCI es un UART full-duplex, que utiliza un formato standard NRZ (Non return Zero), con un bit de start, 8 o 9 bit de datos y un bit de stop.

Estos registros son:

REGISTRO	DIRECCION	DESCRIPCION
BAUD	$102B	SELECCION DEL BAUD
SCCR1	$102C	REGISTRO 1 DE CONTROL DEL SCI
SCCR2	$102D	REGISTRO 2 DE CONTROL DEL SCI
SCSR	$102E	REGISTRO DE ESTATUS DEL SCI
SCDR	$102F	REGISTRO DE DATOS

Tabla 6.1 Registro del SCI

Ejemplo 1

Para la prueba del SCI, se utilizará la misma PC a la cual la pondremos en modo terminal, ejecutando el comando Term del PCBUG.

Programa: Enviar una "A" por el puerto serie

Pasos:

1) Programar SCI
- Baud
- Bit a Tx/Rx
- Habilitar Tx

2) Esperar a que el SCI este listo para transmitir, checando el estado del bit TDRE

3) Enviar dato

4) Proseguir paso 2

El diagrama de flujo se indica en la figura 6.1

PROCEDIMIENTO DE EJECUCION

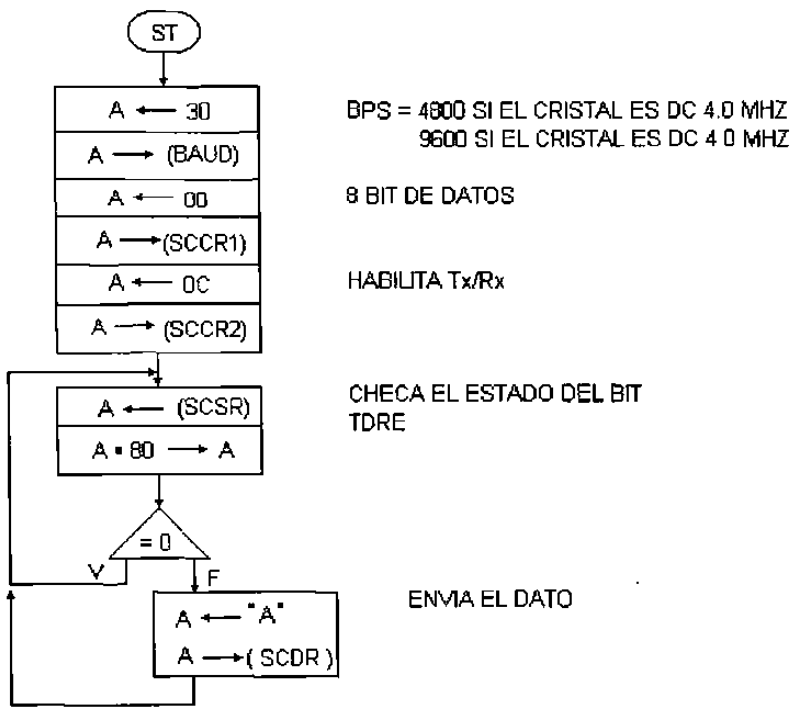


Fig. 6.1 Programa para transmisión de datos

Una vez codificado el programa y capturado en el HC11, usando el comando *MM* o el comando *ASM*, ejecutar el programa utilizando el comando *G* y la dirección del programa, una vez hecho esto, teclear *TERM* para que el **PCBUG** se ponga en modo terminal, al hacer esto en el monitor apareceran "A", para terminar el programar y volver al **PCBUG11** es necesario realizar los siguientes pasos:

1) Restabecer el módulo oprimiendo el boton de *RESET*.

2) Oprimir y la tecla "ESC" para salir del modo terminal y regresar al **PCBUG11**.

3) Teclear el comando "*RESTART*"

PROGRAMAS PROPUESTOS

1) Enviar por el puerto serie el mismo caracter que se reciba a través de este (SCI).

2) Enviar por el puerto serie un letrero cada vez que se oprima la letra "S".

CAPITULO 7

Interrupciones (Práctica No. 6)

OBJETIVO

El Objetivo de este tema es el de apoyar el estudio de las condiociones de excepción como lo es el estado de Reset y las Interrupciones.

INTRODUCCION

Una característica típica de los microprocesadores usados para control, es contar con la capacidad de interrupción.

El concepto de interrupción se refiere, a (suspender/detener) la ejecución normal de un programa, de manera tal que este pueda continuarse en un momento posterior.

Así una interrupción permite al MPU atender o realizar alguna otra acción, por ejemplo ejecutar una rutina llamada de interrupción, que atienda la presencia aleatoria de eventos tales como: alarmas, fallas de alimentación, contabilización de períodos de tiempo, protocolos de intercambio de información con periféricos, etc. y que liberan al programa principal de prestar atención constante a eventualidades mediante supervisión y al programador de la incertidumbre de perder alguno de estos eventos.

Las interrupciones se pueden clasificar basicamente en 4 tipos:

- Reset

- Interrupción por hardware no inhibible (no enmascarable/ no evitable)

- Interrupción por hardware inhibible (enmascarable/evitable)

- Interrupción por software

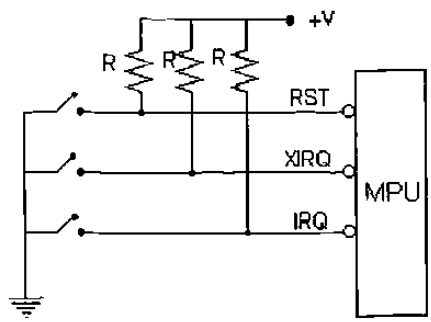


Fig. 7.1 Conexión de interruptores externos a un microprocesador

Al recibir una de estas interrupciones, el MPU carga en forma automática en el contador del programa la dirección de inicio de la rutina de interrupción, a esta dirección se le dá el nombre de vector, y es previamente cargada en ROM en los bytes más altos de memoria reservados para este fin.

7.1 RESET

Aunque la activación de la entrada de RESET provoca una interrupción de la ejecución de un programa, esta entrada no está concebida para la manipulación de entradas y salidas, sino para el restablecimiento de las condiciones iniciales del programa. Al presentarse el RESET el MPU carga en el contador del programa el contenido de las direcciones FFFE y FFFF vector de RESET, que debe contener la dirección de inicio de la rutina de establecimeinto dando así inicio a esta rutina.

La secuencia de inicio se ilustra en la figura 7.2

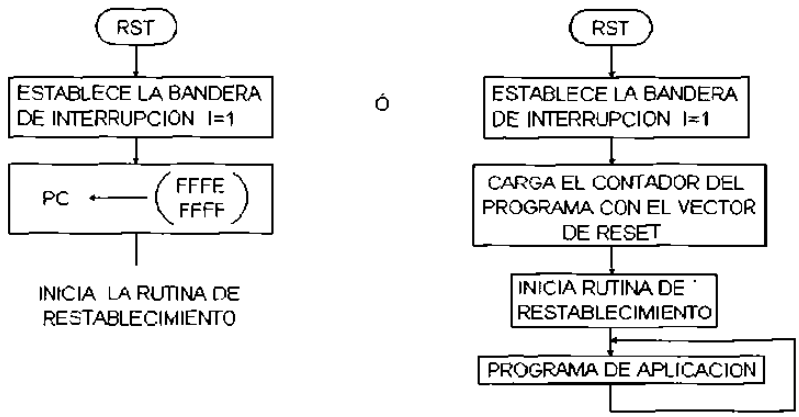


Fig. 7.2 Secuencia de RESET

Cuando se trata de una aplicación específica, es decir el microprocesador operando como un aparato, la rutina de restableciemiento consiste en establecer las condiciones de inicio, para proseguir después con la ejecución del programa de tal aplicación o programa principal.

7.2 INTERRUPCIONES POR HARDWARE Y SOFTWARE

Estas interrupciones provocan que el MPU abandone en forma momentanea el programa principal atendiendo la rutina de interrupción correspondiente cuyo inicio se indica tambien en un par de direcciones llamado Vector de Intrerrupción. En una forma simplificada el diagrama de flujo de la figura 7.3 explica la secuencia de interrupción vía, interrupción por hardware no enmascarable y enmascarable así como la interrupción por software.

Los registros que se guardan en el STACK, contienen el estatus de operación del MPU, durante la ejecución del programa principal antes de la llegada de una interrupción. El contador del programa indica la dirección de la siguiente instrucción a ejecutar una vez regresando de la rutina de interrupción.

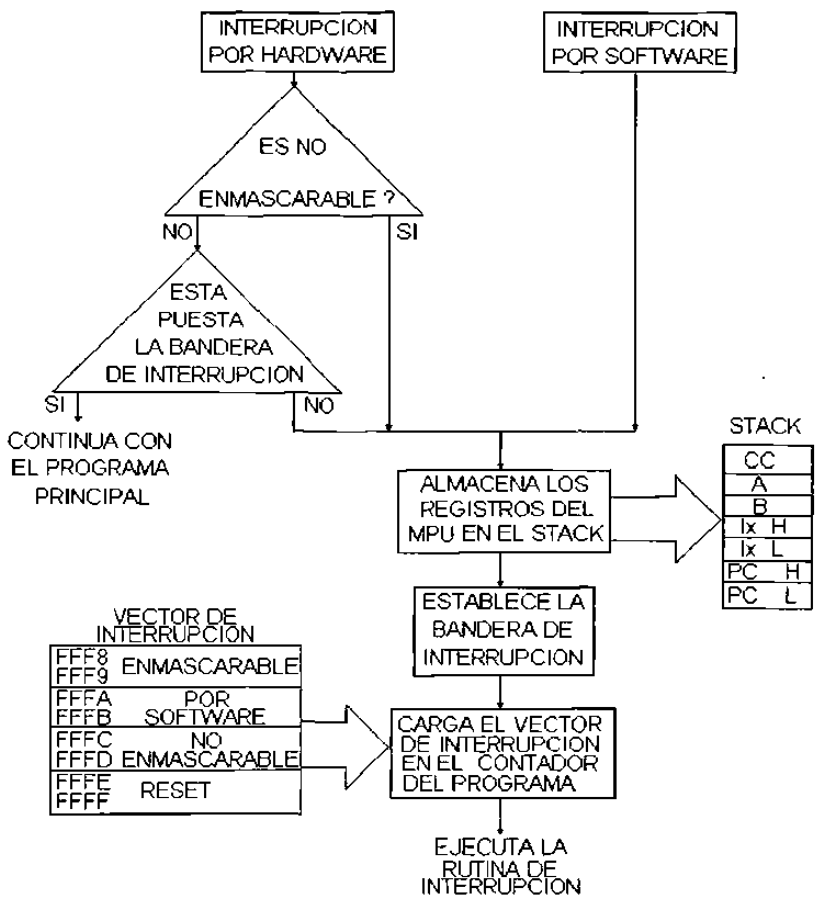
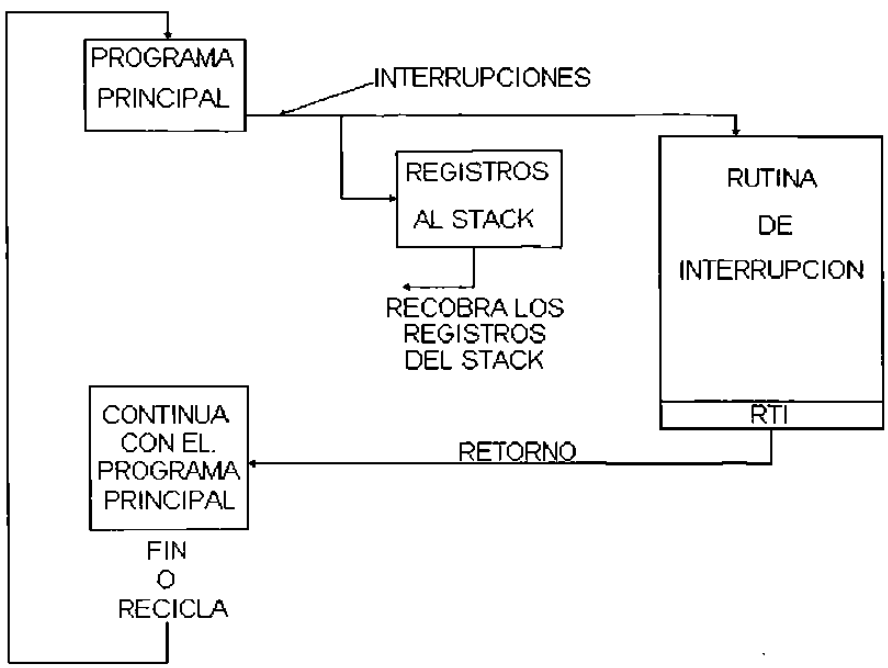


Fig. 7.3 Diagrama de flujo de la secuencia de interrupción

La rutina de interrupción deberá concluirse con una instrucción de RTI, Retorno de Interrupción, con lo que efectúa una secuencia de retorno al programa principal como se muestra a continuación.



CAPITULO 8

Otros bloques del 68HC11
(Práctica No. 7)

8.1 EEPROM

El MC68HC11 cuenta con un bloque de 512 bytes de E^2PROM, localizados en las direcciones de la $B600 a la $B7FF.

Para Grabar/Borrar información en el E^2PROM, se sigue un procedimiento ya establecido, para esto el HC11 cuenta con un registro llamado PROG $103B

Programming Register(PROG)= $103B

0	0	0	BYTE	ROW	ERASE	EELAT	EEPGM	PPROG $103B

Fig 8.1 Registro de control de programación del EEPROM

Byte .- Byte/Other EEPROM Erase mode

Row.- Row/All EEPROM Erase mode

Estos dos bits indican el tipo de borrado a ser empleado, la siguiente tabla muestra la combinación de estos bits:

BYTE	ROW	TIPO DE BORRADO
0	0	BORRA TODO EL EEPROM
0	1	BORRA HILERA (16 BYTES)
1	0	BORRA BYTE
1	1	BORRA BYTE

Tabla 8.1 Opciones de borrado del EEPROM

ERASE.- Erase/normal Control of EEPROM

0= Modo de lectura o modo de programación

1= Modo de borrar

EELAT.- EEPROM Latch Control

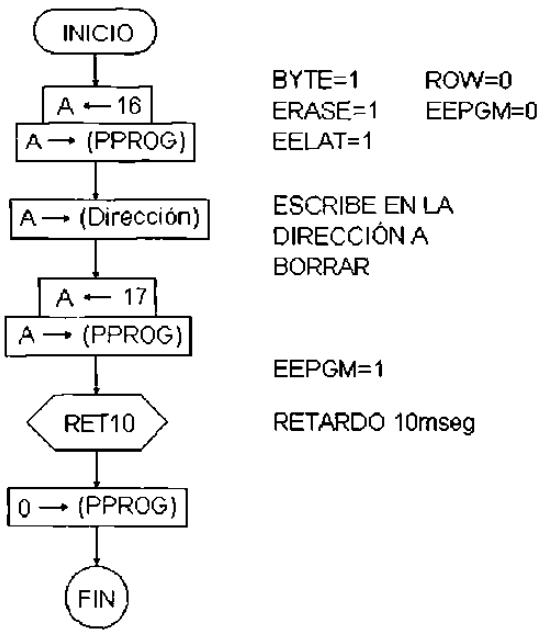
Cuando este bit es cero, el E^2PROM actua como un ROM en el mapa de memoria del MCU. Cuando es 1, actua como si hubiera sido removido del mapa de memoria y puesto en modo de programación, mientras EELAT=1 el E^2PROM no puede ser leido, lo cual implica que la rutina para PROGRAMAR/BORRAR el E^2PROM no puede ejecutarse del mismo E^2PROM.

EEPGM.-EEPROM Programming Voltage Enable

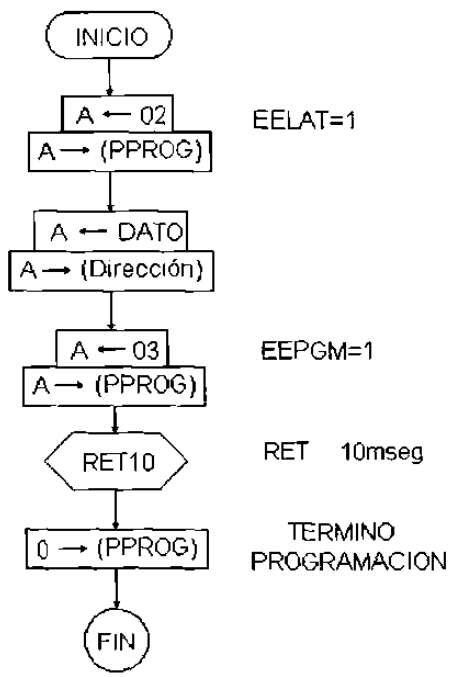
Este bit habilita al Vpp para operaciones de programar o borrar.

Cuando EEPGM=0 Vpp está apagado; cuando EEPGM=1 Vpp está encendido.

Procedimiento para borrar un byte



Procedimiento para grabar un byte



Ejercicio: 1) Hacer un programa que grabe en el E^2PROM la siguiente serie:

0,2,4,6,8,A,C,E,10,.....,1C,1E

2) Hacer un programa que realice un Bulk Erase sobre el E^2PROM

8.2 CONVERTIDOR ANALOGO - DIGITAL

El HC11 cuenta con un convertidor Analógico/Digital de 8 canales de 8 bits c/u, el cual emplea el método de conversion por aproximaciones sucesivas.

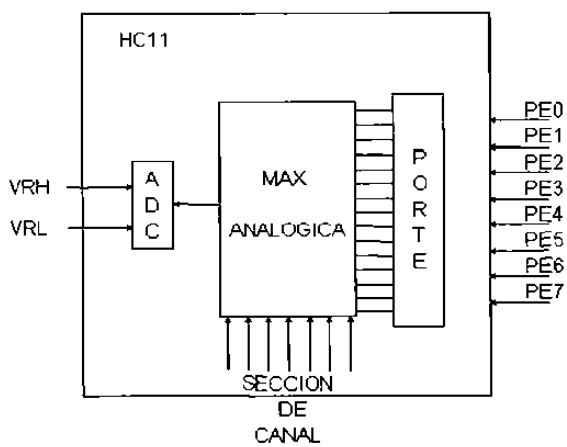


Fig 8.2 Diagrama a bloques del convertidor Analogo/Digital

Registros De Control/status (ADCTL)

CCF=Bandera de fin de conversión

ADCTL $1030

CCF	—	SCAN	MULT	CD	CC	CB	CA

Fig 8.3 Registro ADCTL para operación del CAD

CCF=0 Conversión en proceso.

CCF=1 Fin de conversión.

SCAN= Conversión continua

SCAN=0 El ADC realiza un sólo ciclo de 4 conversiones.

SCAN=1 El ADC realiza indefinidamente ciclos de 4 conversiones.

MULT= Conversión sobre canal o grupo de canales

MULT=0 Realiza el ciclo de 4 conversiones sobre un solo canal

MULT=1 Realiza el ciclo de 4 conversiones sobre un solo grupo de 4 canales una conversión por canal.

CD,CE,CB,CA=Selección de canal.

El bit del Registro Option activa una fuente de voltaje interna, ésta se debe activar cuando

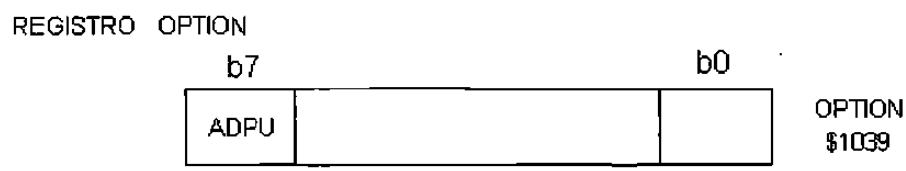


Fig 8.4 Registro Option, bit ADPU

se use el convertidor.

ADPU=0 Fuente desactivada

ADPU=1 Fuente activada

Registro De Resultados:

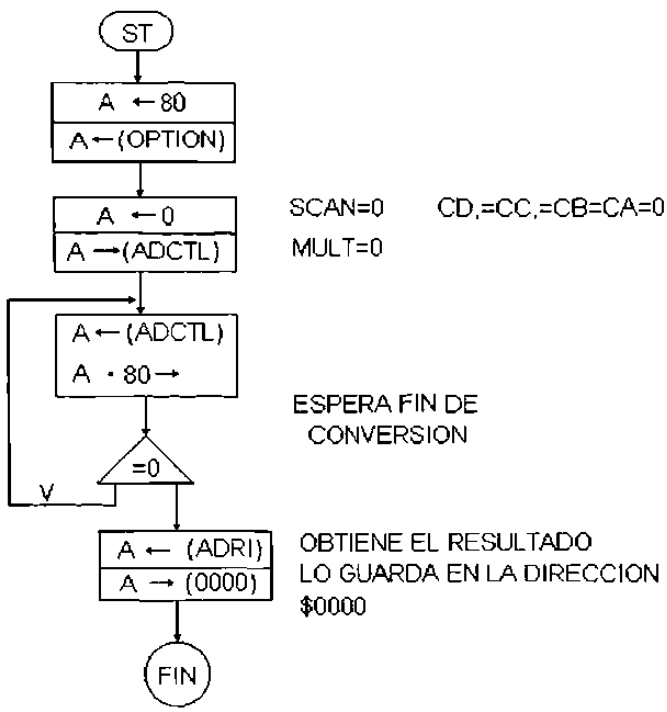
ADR1-$1031

ADR2-$1032

ADR3-$1033

ADR4-$1034

Ejemplo: Convertir la señal analógica que entra por PE0. y el resultado depositarlo en la dirección $0000



PROGRAMA PROPUESTO

Hacer un programa que convierta un grupo de 4 canales (PE4, PE5, PE6 y PE7) y el resultado los deposite en las direcciones $0000, $0001, $0002, $0003.

CAPITULO 9

Aplicaciones
(Práctica No. 8)

OBJETIVO

El objetivo de esta práctica es que el alumno conozca y adquiera la habilidad para manipular los diferentes componentes típicos en las aplicaciones de microprocesadores como instrumento de medición y control de procesos.

INTRODUCCION

Existen diferentes componentes y dispositivos a los que se recurre normalmente en un diseño con electrónica digital ya que solucionan funciones que son típicas en aplicaciones de instrumentación y control de procesos, tales como desplegar e introducir información alfanumérica (display-teclado), movimiento angular y lineal (motor de pasos, motores de C.D.), medición y control de una variable física (control de temperatura).

Cuando el diseño se implementa con bloques de MSI y LSI, existen bloques funcionales que con una mínima o ninguna electrónica adicional se logra implementar la aplicación, cuando el diseño se basa en un microprocesador, este ejecutará programas que sustituye a los bloques funcionales reduciendo de esta manera el número de circuitos necesarios para la implementación del diseño.

En esta práctica se describirá en dos sesiones a algunos de estos componentes, así como un diagrama de idea que sirva como base para que el alumno diseñe programas controladores en detalle.

9.1 SESION 1, MODULO DISPLAY-TECLADO

9.1.1 DESCRIPCION

Este módulo tiene cuatro displays de 7 segmentos multiplexados y un teclado configurado en un arreglo matricial de 4 hileras por 3 columnas.

Las cuatro líneas de selección de los displays se comparten con las hileras del arreglo matricial del teclado, como se muestra en la fig. 8.0.

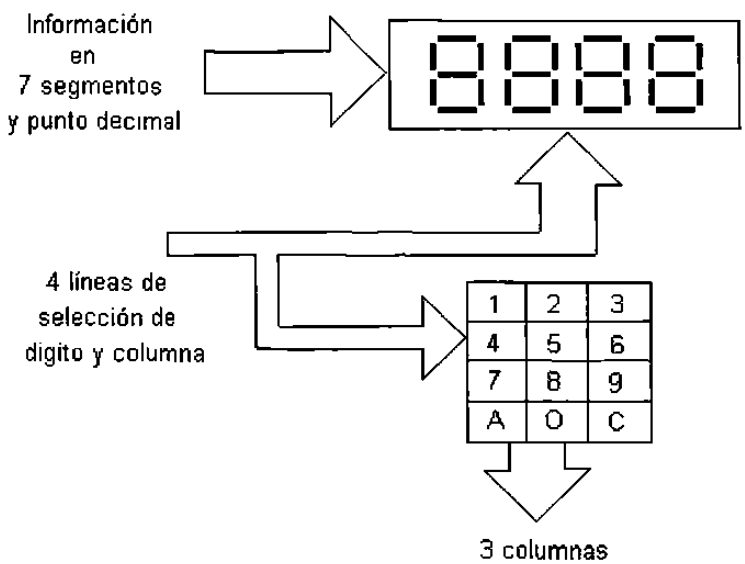


Fig. 9.1 Diagrama a bloques del módulo display-teclado

9.1.2 METODOLOGIA

Para el control de este módulo se requieren de doce líneas de salida y tres líneas de entrada. En la figura 9.1 se muestra el diagrama a bloques de este módulo, en donde se

observa que está preparado para controlarse con 3 diferentes puertos: Puerto B (PB) para el manejo de los 7 segmentos y el punto decimal, el Puerto D (PD) para el barrido o habilitación del dígito y hilera, y el Puerto E (PE) para la lectura de la columna.

El diseño del programa controlador de este módulo se dividirá en 2 partes en donde se tratará por separado el barrido del display y la codificación del teclado, y finalmente se combinarán los dos programas para de esta forma tener el diseño completo.

9.1.2.1 DISPLAY MULTIDIGITO

Estructura del programa

El programa controlador del display consiste en simular los bloques de MSI que solucionarían este problema, que son básicamente un decodificador de línea que habilite un dígito a la vez, y un buffer de cuatro memorias que contengan los datos así como un contador binario de 2 bits que seleccione la línea del decoder y al mismo tiempo direccione al buffer. (Fig 9.2)

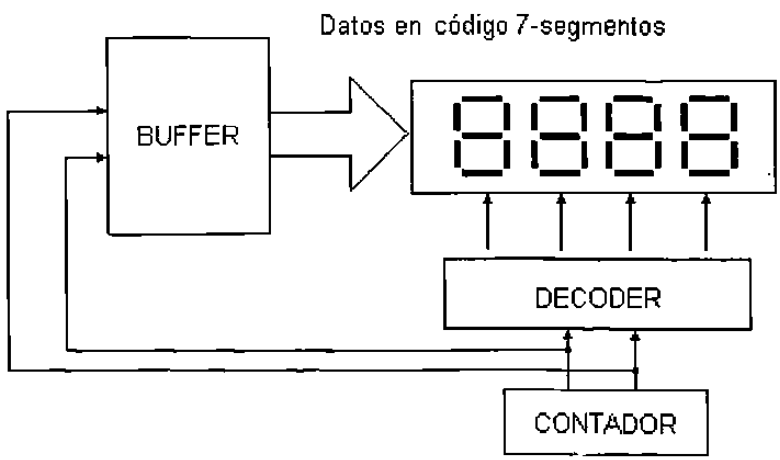


Fig. 9.2 Diagrama a bloques de un controlador de display multidígito.

El diagrama de idea quedaría de la siguiente forma:

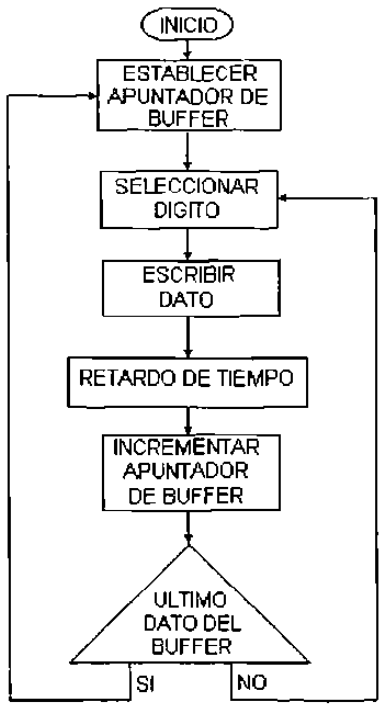


Fig. 9.3 Diagrama de ideas de un programa controlador de display multidígito.

El buffer será un bloque de 4 memorias tipo RAM que contengan el código de 7 segmentos de los caracteres a desplegar, para la selección de los dígitos se implementa un decodificador con 4 líneas en un puerto de salida. El retardo de tiempo se genera de tal valor que la frecuencia de encendido de los dígitos sea tal que al ojo humano parezca como si todos los dígitos estuvieran encendidos continuamente.

Ejercicio 1

Diseñe y codifique el diagrama en detalle:

9.1.2.2 PROGRAMA ENCODIFICADOR DE TECLADO

El teclado consta de 12 interruptores del tipo momentáneo conectados en forma matricial, de ese modo cada tecla tiene su combinación de número de columna y número de hilera.

El programa consiste en seleccionar una hilera a la vez, escribiendo un cero en la hilera seleccionando 4 unos en las demás hileras, de la misma manera de como se efectúa el barrido en el display multidígito, enseguida se leen los niveles en las columnas, cuando una tecla se encuentra oprimida se sensará un nivel bajo en la columna a la que corresponde la tecla presionada, en ese momento se tiene a cual hilera pertenece y para encodificar la hilera se determina la posición del cero lógico en los 3 bits que corresponden a las columnas.

El diagrama de idea quedaría de la siguiente manera:

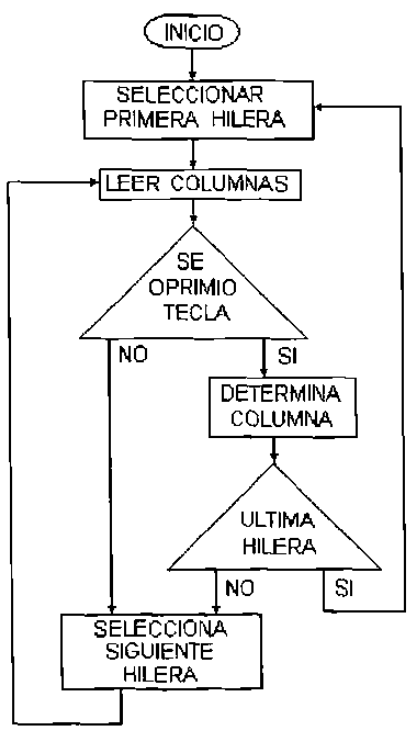


Fig. 9.4 Diagrama de ideas de un programa encodificador de teclado matricial

Este programa arroja como resultado la información del número de columna e hilera de la tecla activada, este dato puede usarse como apuntador de una tabla que contenga el código de cada tecla.

Cuando las teclas son presionadas, éstas tienen un rebote mecánico provocando que pierda el contacto eléctrico varias veces generando con esto que la línea con la que se sensa las columnas oscile hasta que se estabiliza la tecla. Esta oscilación dura unos cuantos milisegundos tiempo suficiente para que el microprocesador lea varias veces esta señal provocándose una situación similar a la que si hubiera oprimido la tecla varias veces. Esto se soluciona si se aplica un retardo de tiempo después de que se detectó la activación de una tecla.

En esta ocasión el problema se soluciona cuando se combine este programa con el programa controlador del display, el lapso de tiempo en que se habilita un display sirve a la vez como retardo de tiempo para así eliminar el rebote.

Ejercicio 2

Diseñe y codifique el programa en detalle:

9.1.2.3 CONTROLADOR DISPLAY-TECLADO

Combinando adecuadamente estos dos programas se tendrá un controlador de display-teclado que cada vez que se ejecute desplegará un dígito, habilitará una hilera del teclado y sensará si alguna tecla fue presioanda.

Ejercicio 4

Elabore un diagrama de idea.

Ejercicio 5

Elabore y codifique el programa en detalle.

9.2 SESION 2, MODULO DE MOTOR DE PASOS

9.2.1 DESCRIPCION

Este módulo está compuesto por un motor de pasos de 15 ^{0}/paso, un sensor óptico para posicionar el motor en una referencia cero o como detector de revolución, un circuito para proporcionar la secuencia de avance del motor. Además se le dotó de una línea para la selección de sentido de giro y una línea que controla la desenergización total del motor para evitar el sobrecalentamiento del mismo.

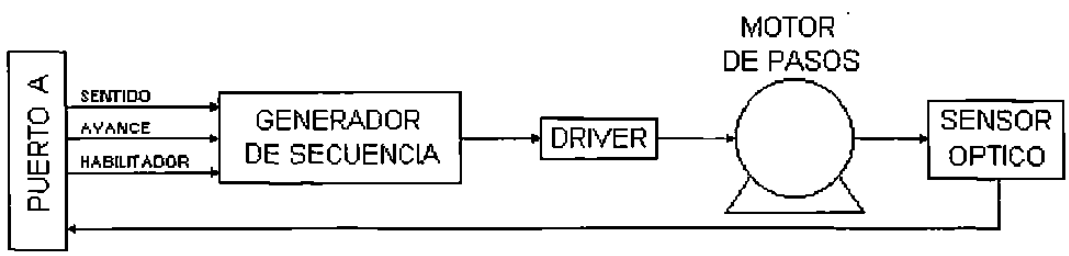


Fig. 9.5 Diagrama a bloques del módulo del motor de pasos

9.2.2 METODOLOGIA

Para el control de este módulo se requieren de 3 líneas de entrada y una línea de salida, la asignación se indica la Tabla 9.1.

PUERTO A	LINEA
PA0	SENSOR
PA3	AVANCE
PA4	SENTIDO
PA6	ENERGIZACION

Tabla 9.1 Asignación de líneas de puerto para control del motor de pasos

Programa de avance de un paso consistirá básicamente en energizar el motor, establecer sentido y generar una transición negativa en la línea de avance y posteriormente desenergizar el motor.

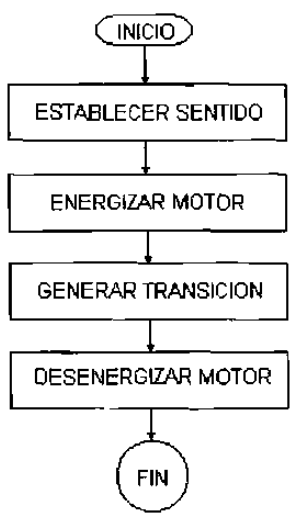


Fig. 9.6 Diagrama de ideas del programa para el avance de un paso al motor.

Si este programa se estructura como una subrutina, en la que datos de entrada serían el sentido de giro y el número de pasos, la operación 1, 2 y 4 solo se ejecutarían una sola vez mientras que la operación 3 se ejecutaría tantas veces sea el número de pasos un diagrama de ideas podría ser el siguiente:

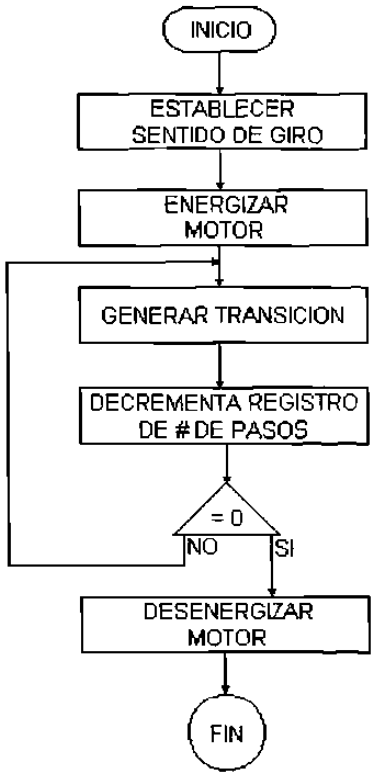


Fig. 9.7 Diagrama de idea de una subrutina de avance de n pasos.

Ejercicio 6

Diseñe y codifique en detalle el diagrama de la figura 9.7

Utilice esta subrutina para implementar los siguientes programas:

PROGRAMAS PROPUESTOS

1.- Utilice la rutina del ejercicio 6 para generar una subrutina en la que el motor avance una revolución completa.

2.- Utilice el programa propuesto 1 para diseñar un programa en el que el motor avance n revoluciones indicadas por los bits 0 al bit 6 del acumulador A, y el bit 7 indicará el sentido del giro con la siguiente lógica.

bit 7	Sentido de giro
0	A favor de las manecillas de reloj
1	En contra de las manecillas del reloj

Tabla 9.2 Convención de sentido de giro para el motor de pasos(Programa 1)

CONCLUSIONES Y RECOMENDACIONES

Es esencial que todas las clases de programación ya sea de superlenguaje o lenguaje de bajo nivel sean complementadas con extensas sesiones de práctica y que además el alumno invierta una buena cantidad de horas fuera del salón de clases y de su horario de prácticas que le establece la escuela para de esta manera llegar a un nivel de entendimiento aceptable del tema y de esta manera poder implementar aplicaciones reales.

Para que el proceso de enseñanza/aprendizaje de la programación de microprocesadores sea lo más rápida y eficaz posible es recomendable que la clase se imparta en un salón-laboratorio en donde el alumno compruebe en forma inmediata los conceptos teóricos que recien acaba de adquirir y además le invierta por su cuenta suficientes horas para alcanzar el fin que se menciona en el párrafo anterior.

REFERENCIAS

1. Bishop Ron ; Basic Microprocessors and the 6800 ; Copyright 1979 ; Hayden Book Company inc., 1978.

2. Clements Alan , Microprocessor Systems Design, DWS-Kent Publishing Company, 1987.

3. García Arregui Macario ; Diseño eléctrico y electrónico asistido por computadora OrCAD/SDT ; Ed. Alfaomega, 1992.

4. Kelly-Bootle & Fowler ; 68000, 68010 , 68020 ; 1ª edición ; The Waite Grove inc., 1985.

5. Leal César A. ; Fundamentos de diseño digital ; FIME ; 9ª edición, Febrero 1991.

6. Leventhal Lance A. ; 6800 Assembly Language Programming ; Osborne & Associates inc., 1978.

7. Malvino; Principios de electrónica ; 4ª edición ; McGraw Hill, Febrero 1991.

8. Mano M. Morris ; Lógica digital y diseño de computadoras ; 1ª edición ; Prentice Hall., 1979.

9. Motorola ; Microprocessor , microcontroller and peripheral data ; Vol I ; 2ª edición; Motorola inc., 1988.

10. Motorola ; Microprocessor, microcontroller and peripheral data ; Vol II , 2ª edición; Motorola inc., 1988.

11. Motorola ; Reference Manual M68HC11., 1990.

12. Savant - Roden - Carpeter ; Diseño electrónico ; 2ª edición ; Addison-Wesley Iberoamericana, 1992.

13. Tokheim Roger L. ; Principios digitales ; 3ª edición ; McGraw Hill., 1990.

14. Webster's New World ; Diccionary of computer terms ; 3ª edición ; Prentice Hall., 1988.

GLOSARIO

Algoritmo	Descripción de reglas y procesos completamente definidos para la solución de un problema o un programa.
Apuntador del Stack	Registro del MPU utlizado para indicar la dirección disponible del Stack.
Baud	Unidad de medida de transmisión de datos. Un bit por segundo.
Bit	Dígito binario, notación de un número en sistema númerico binario.
Buffer	Una de memoria para almacenar datos en forma temporal.
Byte	Grupo de 8 bits operado por una computadora como una unidad.
Código Máquina	Código en binario que un microprocesador reconoce como una instrucción.
Comunicación Asíncrona	Método de transmisión de datos que permite enviar datos en serie en intervalos irregulares precedido cada caracter con un bit de inicio y seguido de un bit de alto.
Comunicación Serie	Transimisión de datos en el cual los bits son transferidos uno por uno.
Comunicación Síncrona	Transmisión de datos en el cual ambos transmisor y receptor usan la misma señal de sincronización.
Diodo Emisor de Luz(LED)	Diodo que al ser polarizado directamente emite luz visibe o invisible
Full duplex	Transmisión de datos en ambos sentidos por líneas independientes.
Lenguaje Ensamblador	Lenguaje de bajo nivel que permite al usuario escribir los programas usando mnemónicos en lugar de códigos numéricos.
Microcontrolador	Unidad de microcomputo para aplicaciones de control.

GLOSARIO (CONTINUACION)

Microprocesador	Sistema basado en una unidad lógica aritmética para procesamiento de datos y control de otros componentes como memoria, dispositivos de entrada /salida todo incluido en un solo circuito integrado.
Mnemónico	Técnica utilizada para ayudar a la memoria humana. Una palabra o nombre que es fácil de recordar. Estatutos que se refieren a las instrucciones de un Microprocesador.
PCBUG11	Sistema monitor de fabricante motorola orientado para desarrollar y evaluar a los microcontroladores de la familia HC11.
Reset	Estado de inicio de un microprocesador y sus componentes.
Stack	Bloque de memoria RAM en el que el microprocesador almacena temporalmente sus registros durante la ejecucuión de una subrutina o un programa de interrupción.

RESUMEN AUTOBIOGRAFICO

Nombre:	José Angel Castillo Castro
Nombre de los padres:	Angel Castillo Castillo Celia Castro de Castillo
Lugar y fecha de nacimiento	Monterrey, N.L. 15 de diciembre de 1964
Grado de escolaridad:	Ing. en Control y Computación Facultad de Ingeniería Mecánica y Eléctrica Universidad Autónoma de Nuevo León
Campo profesional:	Catedrático en la Facultad de Ingeniería Mecánica y Eléctrica de la Universidad Autónoma de Nuevo León impartiendo las clases de Electrónica Lógica III (Diseño con Microprocesadores) y Teoría Electromagnética de 1987 a la fecha. Ingeniero de diseño electrónico en el Centro de Diseño y Mantenimiento de Instrumentos de la misma Facultad, participando en proyectos a la industria y a la misma Universidad, todos ellos basados en microprocesadores de 1985 a la fecha. Jefe del Departamento de Soporte Técnico de la Coordinación de Informática de la FIME de 1996 a la fecha. Integrante del Comité Técnico del Programa de Mantenimiento de Equipo de Cómputo en la UANL de 1989 a la fecha.
Grado que deseo obtener:	Maestro en Ciencias de la Ingeniería Eléctrica con Especialidad en Electrónica
Nombre de la tesis:	Desarrollo de Sesiones de Trabajo para la Programación de Microprocesadores.